Feasibility of Incremental Contract Signing

Michael Stini[*], Daniel Baselt[*], and Martin Mauve[*]

[*] Heinrich Heine University, Computer Science Department, Düsseldorf, Germany

TECHNICAL REPORT TR-2009-001
HEINRICH HEINE UNIVERSITY, DÜSSELDORF, GERMANY
COMPUTER SCIENCE DEPARTMENT

23. 09. 2009

Feasibilty of Incremental Contract Signing

Michael Stini[*], Daniel Baselt[*], and Martin Mauve[*]

[*]Heinrich Heine University, Computer Science Department, Düsseldorf, Germany

Contents

List of Figures

Abstract

In this paper we evaluate the feasibility of *incremental contract signing* on widespread employed mobile devices. Prerequisites for the employment, such as the ability to communicate and to run cryptographic operations, will be motivated and their computational as well as timely complexity will be discussed. We will identify suitable cryptographic algorithms and recommend appropriate key lengths for mobile usage and analyze their performance on real devices. Metrics to rate the scalability will be introduced and applied. The overhead resulting from the communication necessary to conduct incremental contract signing will also be evaluated by the use of different communication patterns. The trade-off between computational and timely savings will be addressed, as the efforts can differ due to different schedules. Taking care of both is crucial for the success of contract signing, because in a mobile environment battery power and time are limited resources. Possible scenarios are sketched based on these measurements and the feasibility of incremental contract signing with respect to these two, above mentioned, factors will be shown. Furthermore, optimizations are presented in form of advanced signing schemes, allowing for an almost 50% gain in efficiency compared to naive schemata.

I. Introduction

We proposed *incremental contract signing (ics)* in [1] as a solution to the problem of signing trade contracts in a fair manner offline, i.e. without the need for any trusted party being immediately involved. Especially the increasing, mobile usage of digital content and the idea introduced by the concept of *Digital Ownership* justifies this approach. The vision of *digital ownership management* (dom) is *not to own the content* – its binary representation– itself, but the rights associated with the desired content. Furthermore it should be possible to trade these rights associated with intangible goods such as digital content at anytime and anywhere. Our statement is, that the (digital) representation of content itself is, due to its widespread availability and ease of access, of no worth. Instead, the services build upon the ownership of rights, which are associated with the content, are of interest as they provide proof of legal usage and added value. In the first place, these rights allow legal users to access the related content by the use of *digital ownership services* (dos). "Conventional online music-stores" lack additional services such as proof of ownership, repeated access in the desired format, and, even more important, the ability to trade and lend these rights. Mobile devices provide enough computational power and storage capacity, to allow ubiquitous usage of digital content. Besides this mobile usage, the demand for spontaneous interactions between participants does exist as users want to show off their digital belongings, use them in the particular manner and finally trade associated rights offline. A detailed description can be found in [2] and [3].

We focus on the offline scenario, where no trusted authority is immediately involved and trading takes place in an ad-hoc manner. Short range communication such as Bluetooth will be employed to search for interesting trade partners in the own proximity and for the entire communication necessary to negotiate and sign the trade contract. Mobile devices should communicate directly with each other instead of relying on an internet connection to access a trusted authority of the digital ownership management to conduct a trade. The aim of this technical report is to prove the feasibility of ics on mobile devices with respect to cryptographic and communication issues.

This report is structured as follows: We will first sketch the big picture in Section II to understand the needs the idea of digital ownership poses on cryptographic algorithms. A brief overview of the system architecture will be given, but the main focus will be on the demands of incremental contract signing. The mandatory computational and timely effort of suitable algorithms and key lengths will be analyzed in Section III. Based on these findings an improvement of almost 50% by adapting the scheduling will be explained in Section IV. Section V is dedicated to the communication overhead resulting from necessary signature handover and the restrictions imposed by a shared, wireless communication channel. An evaluation of the performance will be presented and potential improvements are pointed out. The target platform as well as (freely) available cryptographic libraries will be introduced, compared and rated towards their suitability in the following section. Digital signature algorithms and some widespread used hash functions will be sketched. The mathematical background for a choice of relevant digital signature algorithm will conclude this section. In Section VII real world issues, such as devices, implemented libraries and test-settings will be discussed. Observations and findings made during our test series will be presented. Section VIII contains a conclusion of the insights and results provided by this report. The Appendix contains the statistical plots for chosen scenarios as well as the entire set of histograms.

II. Big Picture

Nowadays, digital content has to be associated with a tangible element, e.g. a CD-ROM, to be traded. The digital ownership management employs a trusted authority that keeps track of the rights which grant access to the content and appropriate services. In order to participate, users and content provider have to register themselves. The latter ones will also register the content they want to offer. Certificates are issued for both, users and content, by the trusted authority stating their authenticity. Public key cryptography will be used for certificates issued by the trusted authority, authentication and for offline contract signing. Initial ownership of rights can be obtained via purchase in a web shop, by redemption of vouchers, give-away promotions, etc. According ownership information will be stored in the database and can be stated in an ownership certificate also issued by the trusted authority. As we assume that devices are not continuously connected to a trusted authority, especially at the moment of an interaction with others. The only information available in such an ad-hoc setting are certificates provided by each user. These certificates have been issued by an authority that all involved user trust and have to be stored on the mobile devices. Now, these devices can disconnect from the trusted authority and strike out for mobile usage. Although all certificates represent a snapshot of the past, *user certificates* and the description of the digital assets, so-called *creation certificates*, are essential to know who you are interacting with and what the subjects of the trade are. While the information inside these certificates can not become invalid, the information inside *ownership certificates*, stating who owns what, can become invalid within an instant when a users *signs a trade contract* for these assets *and hands it over*. Because of this, ownership certificates should only be used as an indicator of ownership, which is similar to the real world, as being in possession of a tangible object does not imply to be its legal owner. These three types of certificates (user, creation and ownership) provide all information necessary to conduct a trade. A user confirms the validity of the ownership certificates, which are relevant to the trade, by signing the trade contract. After the contract has been signed, it has to be submitted to the trusted authority for processing. Processing includes the verification of the attached signatures and contractual liabilities followed by their commitment. In case of proven fraud, prosecution in terms of remuneration, bad recommendations or other sorts of social pressure within the community will take place. Each user has to know her private keys for authentication and contract signing. This knowledge can be verified by anyone using challenges based on the public key, i.e. the information contained in the trusted authority signed user-certificate. See [4] for more details. Please be aware of the fact that the purpose of incremental contract signing is to *enable fair offline trading* within the realm of the digital ownership management, as signing general purpose contracts does not make sense due to missing abilities to enforce the liabilities.

A. Basics of Incremental Contract Signing

The aim of *incremental contract signing* is to enable fair trading while being offline. It will happen that a contract has to be signed by users that neither know nor trust each other. They might be entire strangers which noticed each other by employing interest sensing, which makes use of short range communication as presented in [5]. The following steps, partially relying on cryptographic operations, have to take place: After finding an interesting partner to trade with, priming has to take place by mutual authentication of the participants followed by a presentation of the own belongings and the proof of legal ownership. After the negotiations have taken place, the contractual text is set up and has to be signed by each user. Well known procedures exist for authentication as well as for creation and verification of signatures which also includes certificates [4]. Several approaches tackle the fair exchange of a secret by introducing lots of effort and algorithms dedicated to this special problem, see [1]. In our situation, these secrets would be each users' contract signatures. Our approach allows signing of trade contracts offline in a fair manner by the use of well-known public key digital signature algorithms.

By handing over a signature for a contract, especially when being the first party to do so, an unlimited option to decide if the contract will be submitted and processed is granted to potentially malicious counterparts. The holders of such options can for example wait and search for better offers, play with the market based on the option, or just keep the others from trading as they have signed a contract. The value of such an option depends on the time it can be exploited. By associating an expiration date with every signature, options can be exploited only as long as the inducing signature is valid. This enables a fine-grained control by the signer. Shorter options are of much less value than longer ones, due to the limited time available for negotiation with others, changes of the market and the signer being blocked.

Incremental contract signing relies on a round-based increment of the current signatures' expiration dates. The contract can be submitted for processing as long as a valid signature of each user is attached. Normally, each user's recent signature lasts longer than the one before, pushing the expiration date further. As a result, multiple signatures have to be created and handed over by each user to achieve the desired expiration date of the contract by keeping the risk at an acceptable, low level. Due to this, each user has to receive and verify the signatures of all users before she hands over her next signature. The first signature of each user can not be triggered by this, but has to be handed over by good-will. We assume that no signature expires before all other signatures of a prolongation have been handed over and before the respective user has handed over its longer lasting successor. The determination of the signatures' expiration dates are not subject of this paper, but we will show that a sufficient amount of signatures for *incremental contract signing* can be created and verified on current mobile devices within a reasonable timespan available during an ad-hoc meeting.

Lets have a look at an example: As said before, *incremental contract signing* is build on the idea to restrict the signatures' validity periods to limit the risk each users incurs. For each of these validity periods, regular contracts will be signed by each user with her according signature. For example: A trade contract should be valid for the next 24 hours, but the risk of handing over a 24 hour lasting signature seems too high. If the users agree to sign only contracts that incur a maximum risk of four hours, this 24-hour period will be "split" into six portions, each lasting four hours longer than the one before. The first one lasts only four hour, the second eight, the third twelve hours and so on until the desired duration, 24 hours, have been signed. The ordered handover of these signature grants only an advantage of four hours to a potentially malicious counterpart – because the next, longer lasting signature will only be handed over after the successful verification of all other users' recent signatures. Then each user is in possession of a completely signed contract lasting four hours less than the next one to be signed. At the end a contract lasting for 24 hours has been signed, but each user's risk has been only four hours. The risk can be reduced further by decreasing the validity periods' duration. But then, the number of prolongations g increases. In our first approach, a single signature of each user is needed per prolongation. Now, a trade-off between the incurred risk and the number of prolongations does arise and has to be adjusted. If the risk should be smaller the more prolongations are necessary – resulting in an higher effort for signing. Vice versa, the risk becomes higher the less prolongations are desired. To reduce the risk by a factor of $1/g$, the effort increases by the factor g.

B. Mandatory Cryptographic Operations

The focus will now be on the cryptographic operations that are needed to realize the functionality of the digital ownership management and especially for *ics*. Short range communication such as Bluetooth will be analyzed in Section V. Concerning cryptography, it has to be distinguished between

a) operations that have to take place in real-time, e.g. contract signing or signature verification, where others are waiting for the result of the operation, and
b) operations that can be accomplished at anytime before and laid in stock for later usage, e.g. key pair generation.

Operations that take place at the trusted authority are of no concern as sufficient computational power as well as libraries are available[1]. We focus on the question, if the computational power of mobile devices and the performance of libraries as well as desired algorithms suffice the needs.

[1] Issues such as scalability and protection against (distributed) denial of service attacks are not subject of this paper.

Offline trading relies on two sources of information,

1) the certificates signed by the trusted authority and
2) the information assembled and signed by the users.

Both have to be handed over and verified while negotiations and contract signing takes place – just-in-time. Signature creation and verification are mandatory to allow a mobile device to participate. Although these operations rely on public key cryptography, which is slower than symmetric cryptography, they have to be possible within an acceptable timespan despite the very limited computational power available on the mobile devices. The main focus of this technical report will be on the evaluation of the two crucial operations: creation and verification of signatures.

We assume that the trusted authority is capable of all operations. In contrast, mobile devices have to provide signature creation and verification as the least common denominator. Optional, but desired functionalities are additional services such as the creation of key pairs as well as encryption of the contract with the trusted authority's private key. Normally the private key should never leave the device it is created and used on. In this case the mobile device has to be able to generate cryptographic key pairs, register the public key with the trusted authority and prove the knowledge of the private key by responding to a challenge based on the public key to be registered. Otherwise the user's key pair has to be created by the trusted authority and transferred to the device. Encryption of the contract with the trusted authority's key is necessary to ensure privacy as only the trusted authority can encrypt the contract during its processing. This procedure should be used if users that are not directly involved in a trade get hands on the contract, e.g. in case of chained offline trading or if the contract will be submitted by users not participating in the trade. The following list provides a summary of all operations; the optional ones are marked with *:

- *at the trusted authority (TA)*
  - creation of a strong (root) key pair for the trusted authority,
  - creation of key pairs for the users,
  - creation of (user, creation, ownership) certificates,
  - decryption of contracts encrypted with the TA's public key,
  - verification of signatures generated by users and
- *at the mobile device (MD)*
  - creation of key pairs*,
  - verification of the certificates issued by the trusted authority,
  - creation of signatures for contract signing,
  - verification of signatures generated by users and
  - encryption of contracts with TA's public key*.

III. Cryptographic Costs of ICS

The aim of this analysis is to estimate the cost of *incremental contract signing* with emphasis on the cryptographic operations. Costs will be evaluated with respect to the computational power needed to create and verify the signatures and the time needed to do so. Although both aspects seem to be linearly related, this does not hold for all settings as different schedules can be employed. Because it is not possible to determine the number of operation to express the computational costs, they will also be expressed by the time the operation needs. Based on fundamental findings, which will be described throughout this section, the feasibility for a given settings can be easily evaluated.

Therefore, we will

1) determine the number of signature creations and verifications necessary as well as
2) derive the cost of these operations and
3) introduce metrics to allow for comparison considering the number of involved users and the desired cryptographic strength
4) and finally calculate the time span needed to sign contracts by for the proposed schedules.

A trade-off between the number of users n, the desired cryptographic strength and the time available for signing d_{sign} does exist. We consider only reasonable parameterization; excluding both extreme courses: The one with minimal effort, where each users' first signature lasts until the desired expiration date of the contract and the unsuccessful one with maximal effort as useless signatures with too short validity periods are exchanged. The minimal effort is de facto *naive* contract signing as only a single, long lasting or an even unlimited valid signature of each user is needed. The latter one with maximum effort will happen in case of wrongly-chosen parameters or it has been caused on purpose. Then it is some sort of denial of service attack, but with high costs on both sides. Nevertheless, the subject is reasonable and therefore successful parameterization and potential improvements by examining the computational as well as the timely cost if n users are involved. This means that the number and cost of cryptographic operations will be determined and time necessary to do the processing and to wait for the signatures to be handed over in the chosen setting. Two scenarios are explained in the following sections.

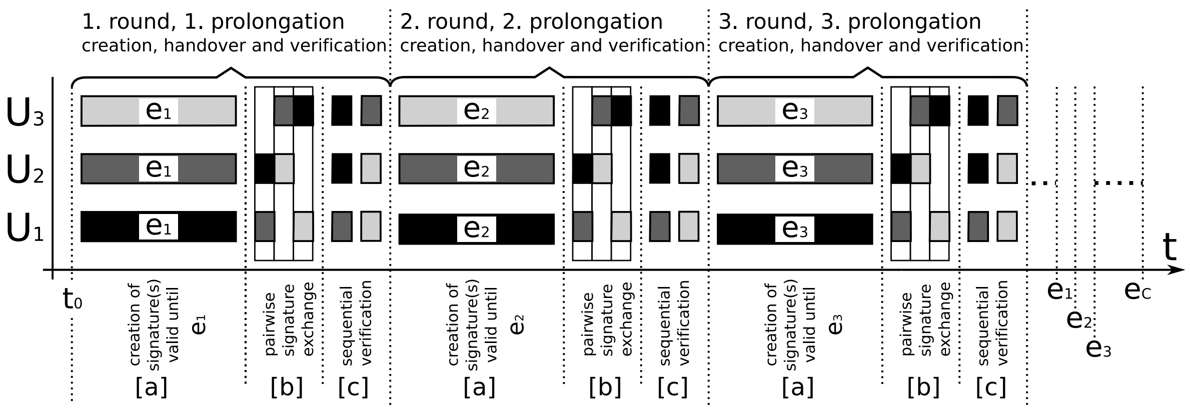
A. Parallelized Schedule

The first schedule presented here is straight forward and called *parallelized* as maximum parallelization of operations is employed and the signatures exchanged during one round are dedicated to the same expiration date. A round is defined as the handover of n subsequent signatures, the longest lasting signature of each user that has not been handed over before.

The calculation of the expiration dates of every signature is possible according to the chosen risk assessment strategy before-hand [1]. Each round consists of three phases which can be easily identified in Figure 1:

[a] (parallelized) signature creation,
[b] their pairwise (sequential) exchange and
[c] finally their (sequential) verification.

The exchange of signatures between all users has to be sequential, because the communication channel is shared and the communication itself is unicast. A similar restriction exists for the verification of signatures due to single core hardware of the processors used for cell phones. Every user has to hand over his signature to all others. If n users are involved, each user has to hand over her own signature $n-1$ times and to verify $n-1$ signatures of the other users. The synchronization of these three phases results from the fact that a user will handover her next, longer lasting signature only after she has received and verified all other users' recent signatures. As a result, the time needed to perform incremental contract signing is dominated by the slowest device. Obviously, the computational cost will be the same for each user as they have to process the same amount of signatures. Figure 1, which will now be described, provides a sketch of the first three rounds in such a setting where three users are involved. The users can be distinguished by different shapes of gray. The three phases during each round can be easily identified: First, each user creates the signature valid until the expiration date, e_j, associated with a particular round [a]. j refers to the number of the prolongation, which is in the parallelized setting equal to the number of the round. Then a pairwise handover of the signatures takes place, one pair after another which is indicated by the vertical slots and the signatures that are exchanged [b]. Each user has to verify the signatures of the other users after the exchange of all signatures [c]. The process starts over by creating signatures for the next expiration date if the validity all signatures has been checked successfully. As soon as the expiration date e_i is equal or greater than the desired expiration date of the contract e_c, incremental contract signing succeeded. In this parallelized schedule, a prolongation, i.e. the increase of the signature's expiration date, takes place once each round. But, the expiration date will only be prolonged if each user has handed over, received and verified the signature and accepted the expiration date of all other users' signatures. Therefore, the number of prolongations g is equal to the number of rounds c.



1. round, 1. prolongation
creation, handover and verification
2. round, 2. prolongation
creation, handover and verification
3. round, 3. prolongation
creation, handover and verification
U3
U2
U1
e1
e2
e3
t
t0
creation of signature(s) valid until e1
pairwise signature exchange
sequential verification
creation of signature(s) valid until e2
creation of signature(s) valid until e3
[a]
[b]
[c]
e1
e2
e3
ec

Fig. 1. Parallelized Schedule, 3 Users, 3 Rounds, 3 Prolongations.

Our feasibility analysis is based on such a setting. We refer to the cryptographic operations that are directly involved in incremental contract signing: *creation* and *verification(s)* of a signature, as *signature processing*. $n-1$ signatures, one from each user, have to be processed by every other user during each prolongation. While a signature can only be created once per prolongation and user, its verification has to be done by all other, $n-1$, users.

In this context, the time needed to hash the text to be signed and to be stored is small enough to be neglected. The overhead caused by the signature exchange will be evaluated separately in Section V. Further we assume that key generation has already taken place.

The analysis starts with the cost estimations for each user. We assume that g prolongations are necessary. Thus, the number of signatures each user has to create is equal to the number of prolongations g. Because each user has to verify the signatures of all other users during every round, $g(n-1)$ verifications have to be done per user. Parallelization of the signature creation is possible due to the determinism of the signatures' expiration dates. After this introduction we will distinguish precisely between timely and computational cost. If more than two users are involved, sequential verification takes place at each user, as shown in Figure 1. During each *round*, a single signature has to be handed over by each user to all other users; in this example in a round robin manner: $U_1, U_2, U_3, U_1, \ldots$ and so on. Furthermore, each round is dedicated to a single prolongation: c rounds with $c = g$ are necessary. The handover between pairs of users is realized as a bi-directional exchange, i.e. each of the two users hands her signature during a single communication over shown in the Figure 1. The colored blocks in each communication slot indicate the signatures which have been received. Parallelization of the users' actions are possible, which results in a timely cost that is equal to the computational one per user. The overall computational costs are n-times the cost per user. Although some devices might be faster, they will be "synchronized" by the next handover as normally no user wants to incur more risk than any other. Because of this, the overall time needed is determined by the slowest device.

To analyze the cost, both operations – signature creation and verification – have to be considered separately as *1)* their execution times and *2)* their number of executions is different in a scenario with more than two users. The execution time depends on the employed algorithm, the key length and the device. Algorithm and key length have to be chosen carefully to achieve desired level of security under the given circumstances, such as the value of the traded assets, time available for contract signing, desired contract expiration date and multiple usages of the key pair. Recommendations on the key length can be found in Section VI-G. Detailed measurements, which are the substantiation for the following argumentation can be found in the Appendix.

Our cost analysis consists of the following five steps:

1) representation of the cost for verification by the cost for signature creation *(inner ratio)*,
2) representation of the cost of different key lengths by the shortest one *(outer ratio)*,
3) combination of inner and outer ratios,
4) application of real run-time measurements for the key lengths' used as unit and
5) summarization.

B. Inner Ratio

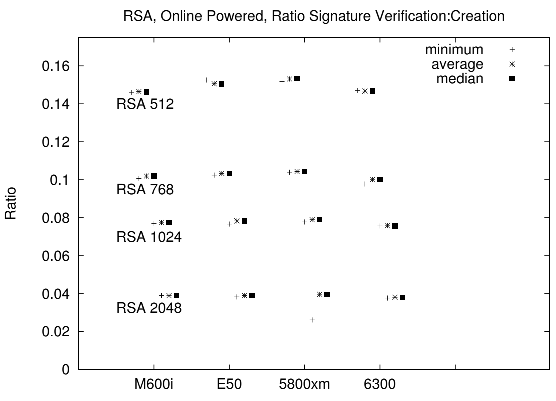
Signature processing has been introduced as the combination of the two signature centric operations, which have to be done just-in-time: *creation* and *verification*. These are the building blocks for the estimation of the costs because they take place during each round by every user. The timely as well as the computational costs for the processing of a single signature of each users is composed of the cost for signature creation and, as all users create signatures, the verification of all other users' signatures. This is summarized in Equation 1.

$$|signature\_processing| = |creation| + (n-1)|verification| \tag{1}$$

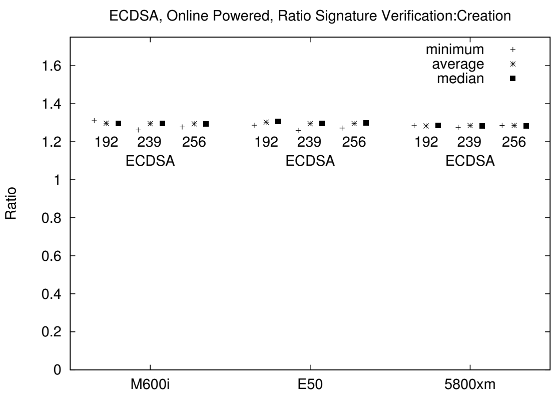
To simplify the analysis it is useful to determine the ratio – if any exists – of both operations' run-time. Instead of dealing with the costs for creation *and* verification, the cost of processing could be expressed by means of signature creation. Additionally, this would allow a better understanding of the scalability as the cost depends on two variables which have different impact: the number of prolongation g and involved users n. Then, for further analysis only a single run-time measurement, the execution time of one signature creation is necessary. Although feasibility studies would be possible by the direct usage of run-time measurements, our approach allows for generalization. As we will see later on, it is also a useful to rate the scalability and the suitability of different communication patterns.

We refer to the *ratio of signature verification to its creation* as *inner ratio* because it is specific to the considered algorithm and the key length. Both operations are deterministic processes for a given key. Because of this, there has to be a correlation between the efforts needed to create a signature and its verification. In contrast, RSA key generation has a non deterministic run-time as it relies on picking very large numbers and testing them to be prime; which might fail and the process has to start over. A mathematical analysis seems to be possible by estimating the bit pattern of the exponent and derive how many exponentiations are necessary on average. But, an examination of the run-time measurements on real devices have to be done anyways as implementation and device specific issues might influence the theoretically derived results. The ratios of the minimum, average and median run-times have been derived from these measurement and can be found in Figure 2 for several key lengths and devices when being powered, online and idle – which means processing nothing except our speed measurements. The run-times of signature creation and verification have to be linearly dependent, i.e. these ratios should be about the same on all devices, for the above introduced assumption to hold. Please notice the difference of the vertical axis, which is a factor of 10.

Let's first have a look at RSA, whose ratios are plotted in Figure 2(a). The abbreviations used, refer to the devices the majority of our test runs took place, the Sony Ericsson *M600i*, the Nokia *5800xm*, *6300* and *E50*. More details about these device can be found in Section VII. It can clearly be seen that for each key length a specific ratio does exists. This is reasonable, because the complexity of RSA verification and creation differs. More details can be found in Section VI-F.1. The plots illustrate that the proposed correlation between signature creation and verification exists, i.e. that the ratio does not depend on the device, but only on the key length and algorithm. The minimum,



(a) RSA 512, 768, 1024, 2048.



(b) ECDSA 192, 239, 256.

Fig. 2. Inner Ratio of Run-times, Signature Verification:Creation, RSA and ECDSA

median and average value have been considered and their *inner ratio* has been calculated – which are nearly identical. Please be aware of the fact that we do not refer to the minimum, median and average of the ratios, but to the ratios of the minimum, median and average values. Due to this, the "minimal" ratio is for some devices slightly larger than the average and medium ratio or as in a single case about 30% smaller. The ratio of the maximums have not been considered as they varied too much to provide any insight in this context. It can be seen that RSA verification is – due to the chosen exponent of $2^{16}+1$ – quite fast compared to signature creation. The usage of such an "inexpensive" exponent is recommended by X.509 [6] and harmless as long as certain requirements are met. In case of ECDSA, which is shown in Figure 2(b), all ratios are approximately the same. Neither the device nor the key length seem to have any influence. Which is in contrast to RSA, where the ratio clearly depends on the key length.

All this are strong indicators that the assumption of a device, operating system and virtual machine independent *inner ratio* is true. Sufficient indicators and information have been collected to allow the exploitation of this to ease cost analysis and to decide on the suitability of certain scenarios. Now, it is possible to express the cost of incremental contract signing with the run-time of signature creation as unit. The estimated ratios, summarized in Table I, are based on the medians of the particular setting's measurements.

TABLE I
INNER RATIO OF MOBILE DEVICES, SIGNATURE VERIFICATION:CREATION.

Device	RSA512	RSA768	RSA1024	RSA2048	ECDSA
M600i	0.1429	0.0994	0.0777	0.0394	1.2913
E50	0.1333	0.1000	0.0781	0.0389	1.2952
5800xm	0.1458	0.1032	0.0784	0.0396	1.2842
6300	0.1435	0.0978	0.0754	0.0379	∅

As stated before, there are two aspects to consider: the computational and the timely cost. The computational cost per user is equal to the timely one. The overall timely cost is, due to parallel processing, equal to the one per user, the overall computational one is n-times the one per user.

If, for example, RSA512 is employed, signature processing would "cost", according to Equation 1 each user per round approximately

- $1+0.14 \quad = 1.14$ signature creations in a two user scenario,
- $1+2\cdot 0.14 \quad = 1.28$ signature creations if three users are involved.

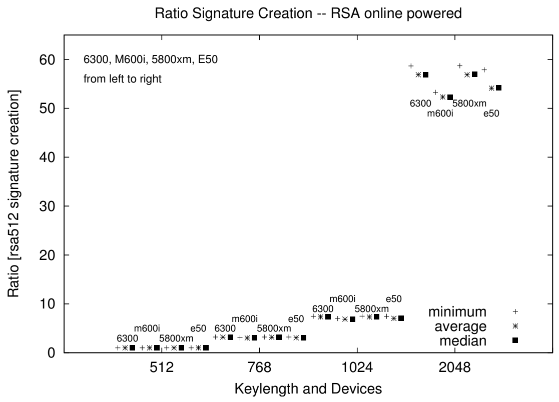
The longer the key is, the cheaper the verification becomes. Employing RSA1024 would result in costs of 1.078 in a two user and 1.156 in a three user setup. Verification of signatures using ECDSA is, in contrast to RSA, slower than their creation – but independent of the key length. Therefore, signature processing by the use of ECDSA costs for every key length.

- $1+1.29 \quad = 2.29$ signature creations in a two user scenario,
- $1+2\cdot 1.29 \quad = 3.58$ signature creations if three users are involved.

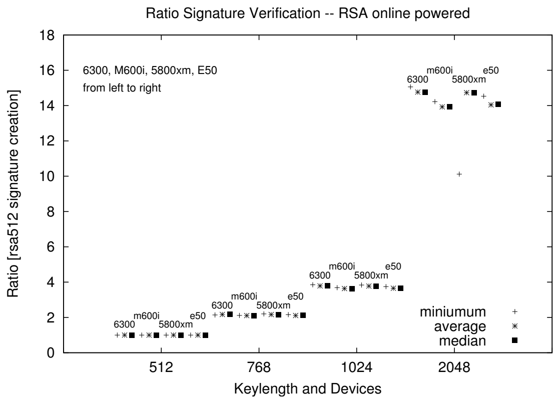
These are important information as one signature processing has to be done per user during each prolongation. In case of ECDSA the number of users does not scale well because signature verification, as the more expensive operation, has to be carried out more often. RSA verification is compared to creation – due to the suitable chosen exponent – quite fast and it becomes faster the longer the key is. This effect is due to the different complexity of signature creation and verification [7]. But, so far only the *inner ratio* has been determined, not allowing for a direct comparison of the different algorithms and their key lengths. This will be subject of the following section.

C. Outer Ratio

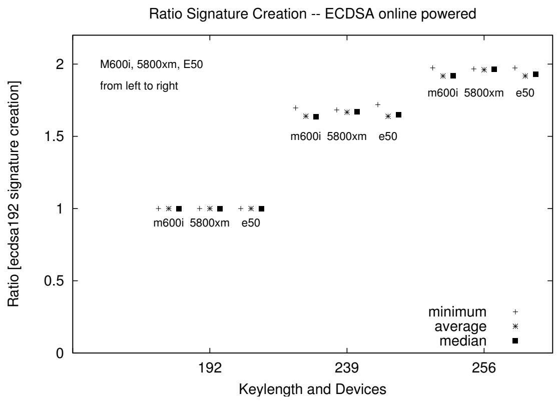
So far, the operations within a certain key length have been compared and a respective ratio has been derived. The use of such a ratio allows for a very fast and effective comparison. We will now choose a well-known reference, compare the executions times and derive the ratios to express their difference depending on key lengths. We refer to this as *outer ratio* because it puts different key lengths and thereby different levels of security, in relation to each other. RSA512 and ECDSA192 in the *online powered setting* have been chosen as reference for the respective algorithm. Again, only creation and verification of a signature are of interest and have to be estimated for each device. This time, individual plots for the ratios of each of these two operation are provided in Figure 3. The ratios have been derived from the same set of measurements. As expected, the ratios seem to be device independent. A slight deviation between devices becomes noticeable for the longest key lengths evaluated, RSA as well as ECDSA. This can clearly be seen in case of RSA2048 where the difference between devices is $\approx \pm 6\%$, which is tolerable as these ratios should only enable an approximation. These deviations seem to be device dependent as they appear for both algorithms, RSA and ECDSA, the same way: The E50 and the M600i have lower ratios than the 6300 and the 5800xm, which means that they are slightly faster. Table II summarizes these findings by the use of an average value across devices. To ease understanding, the inter-algorithm comparison has not been included in this table, but run-time measurements for the recent version of the employed Bouncy Castle [8] cryptography libraries can be found in Table III. These measurements are just a snapshot of the actual libraries and employed hardware. Therefore, improvements of the implementation and the use of faster as well as optimized hardware have to be continuously evaluated. Every single aspect can have a major impact on such an inter-algorithm ratio[2].



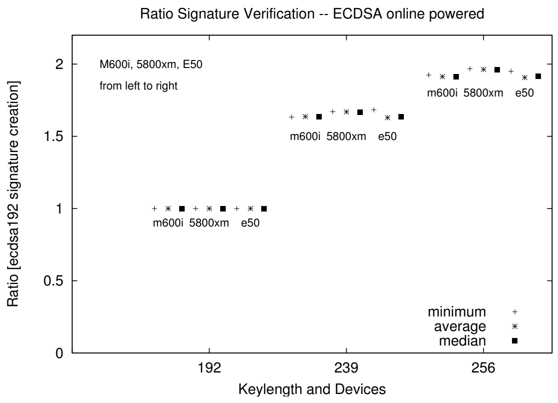
(a) Outer Ratio, RSA, Signature Creation.



(b) Outer Ratio, RSA, Signature Verification.



(c) Outer Ratio, ECDSA, Signature Creation.



(d) Outer Ratio, ECDSA, Signature Verification.

Fig. 3. Outer Ratio – RSA and ECDSA – Signature Creation and Verification.

TABLE II
AVERAGE OUTER RATIO OF MOBILE DEVICES.

	RSA512	RSA768	RSA1024	RSA2048	ECDSA192	ECDSA239	ECDSA256
RSA512	1.00	3.18	7.19	52.39	-	-	-
ECDSA192	-	-	-	-	1.00	1.64	1.94

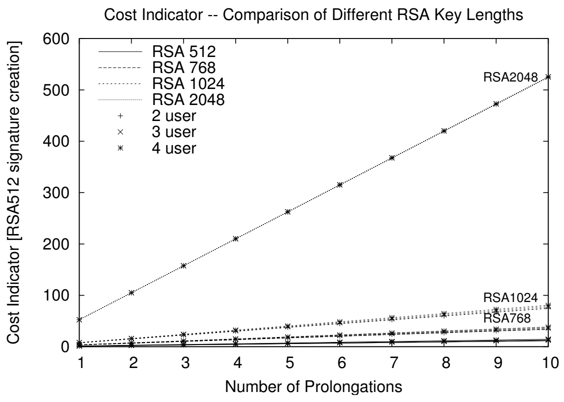
D. Overall Ratios

By combining both, *inner* and *outer*, ratios it is possible to use the run-time of RSA512 resp. ECDSA192 signature creation as unit for cost estimation of longer key lengths. Combination takes place by the multiplication of *inner* and *outer* ratio; their product is called *cost indicator*. They are plotted in Figure 4 for different key lengths and number of involved users to enable the direct comparison of their cost and visualizes their impact. Graphs are provided for settings with two to

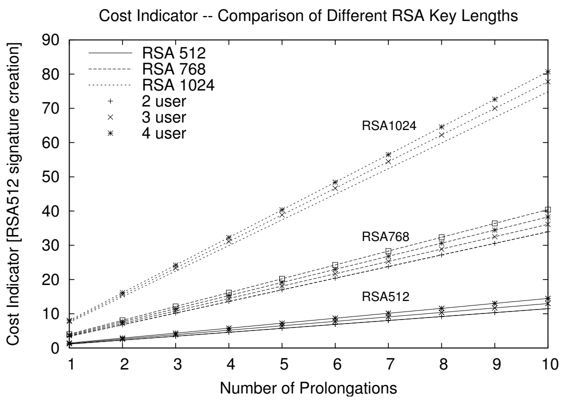
[2]The inter-algorithm ratio will not be considered in this paper.

four users and up to ten prolongations. Please remind, that we still provide only the cost indicators, i.e. relative numbers. Real measurements are subject of the next section. Two plots, Figure 4(a) and 4(b), are provided for RSA. The first plot contains all RSA key lengths to provide an overview, allowing for a comparison among similar ECDSA and RSA key lengths, and the second one without RSA2048, because RSA2048 signature processing results in large values rendering the other, lower ones almost indistinguishable. It can be seen, that RSA does, due to the inexpensive verification, scale well with the number of users. Because of this, the graphs for the different RSA key lengths are clearly separated, but grouped by the number of participants, i.e. key length is the dominating factor. In contrast, the cost indicators for ECDSA do neither cluster by key length nor by the number of involved users, i.e. their graphs are mixed up. Choosing a longer key length has the same implication for the cost as adding another user and vice versa. Three factors can be pointed out: *1)* Verification is more expensive than creation and has to be done more often for $n \geq 3$, *2)* inner and outer ratios are about the same and *3)* the outer ratios themselves do not differ that much.

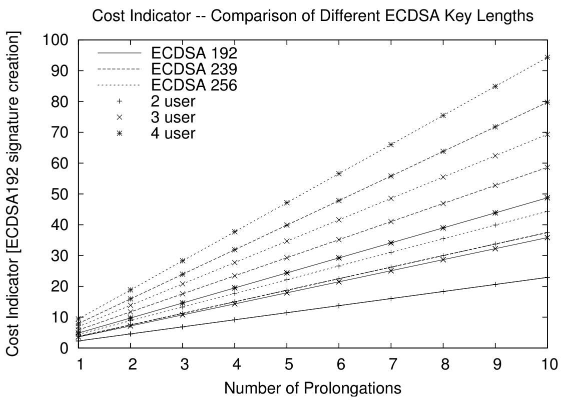
It turned out, that RSA scales well with an increasing number of user and that the desired key length can be chosen independent of the number of users. Whereas this an issue with ECDSA, because adding one user results already in a degradation of security, because a shorter key length has to be chosen if the time available for contract signing is limited.



(a) Cost Indicator for RSA 512-2048.



(b) Cost Indicator for RSA 512-1024.



(c) Cost Indicator for ECDSA.

Fig. 4. RSA and ECDSA – Cost Indicator

To determine the real cost for each combination of device, algorithm and key length, the run-time for the basic operation, signature creation, has to be determined. Of course, real measurements have been used to derive the findings presented so far. These depend on the load status of the device and will be subject of the following section.

E. Real Measurements

So far, all comparisons are relative to the cost of the two chosen units, the run-time of signature creation by the use of RSA512 and ECDSA192. Table III provides these two missing measurements to enable calculation of the costs that arise in realistic settings. Two out of five examined scenarios have been considered here: The most probably fastest one, where the device is running only our calculations while being power-supplied and the other one with un-powered devices and their "specific full-load", i.e. media-player and if available internet access by the use of WLAN and GPS turned on. The run times for both settings are listed together with their difference as percentage for each device and setting. We decided to use these two in our opinion extreme, but realistic settings. Obviously, it depends on the load of the mobile phone as the run-time increases for the E50 by as much as 83%, for the other three devices of about 40% if load is imposed on the device. The E50 seems to have massive problems with the task scheduling when content is played back in the foreground and speed tests are run in the background. The increase by 83% has been caused by audio playback, while video playback almost stopped our test application. The RSA2048 key generation needed multiple hours instead of the usual couple of minutes. Thus, we noticed the E50 measurements, but do not consider them as we refer to an assumed average increase of the run-time of about 40% from an "idle" to a full-loaded device. Nevertheless, the capabilities and strange behaviors of employed devices have to be figured out before starting mission critical operations or releasing software for such affected devices.

Three important facts have to be pointed out:

1) The major difference between the speed of the chosen RSA and ECDSA key lengths, which can be expressed by an *inter-algorithm* ratio. This factor is listed in the last column of Table III, but no further investigation will take place. The factors for these two settings are about the same for every device, but they strongly rely on the device itself. To rate the different computational power of devices an *inter-device* ratio could be introduced.

TABLE III
SIGNATURE CREATION. RUN-TIME[MSEC] ON VARIOUS DEVICES AND SETTINGS.

	RSA512			ECDSA192			RSA : ECDSA	
Device	idle power	increase	full-load unpowered	idle power	increase	full-load unpowered	idle powered	full-load unpowered
M600i	55.6	≈35 %	74.0	2391.0	≈31 %	3142.0	1 : 43.00	1 : 42.46
E50	45.3	≈73 %	78.5	2023.0	≈83 %	3711.0	1 : 44.66	1 : 47.37
5800xm	48.7	≈37 %	67.0	1706.5	≈45 %	2489.5	1 : 35.04	1 : 37.15
6300	115.3	≈39 %	160.5	∅	–	∅		

2) Although our test series ran only on a limited set of devices, the difference between them became obvious with the 5800xm able to compute ECDSA only about 35 times slower than RSA compared to the E50 and M600i with a factor of about 45. Again, the cell phone has to be regarded as a black box, not knowing where the difference comes from.
3) Not only the relation between certain measurements are different, but also the basic run-times differ by a factor of about 2. The 6300 is only half as fast as the other devices, although a processor of similar speed is employed.

F. Summary

A lot of important information have been gathered during our test series, while evaluating the performance of devices, algorithms and operations. Furthermore, their relations have been evaluated and exploited. The existence of an *inner* and *outer ratio* has been documented and successfully used. Typical usage scenarios have been classified and their impact on the performance have been recorded for later usage. Different characteristics and available computational power of devices have been identified.

The availability of strong cryptography as well as its feasibility has been shown. It turned out that the clear advantage of RSA is its ability to verify signatures fast compared to their creation, which has been made visible by the (small) inner ratio. A fast verification allows to increase the number of users with almost no impact on the performance because the cost for signature processing are kept low. In case of ECDSA, signature verification is more expensive than creation, not allowing for scalability with respect to the number of users.

Additionally, the creation of signatures is about a factor of 35 to 45 faster compared to ECDSA. But, this ratio seem to be device dependent, i.e. relate to different hardware, operating system and virtual machine combination as device specific adaptations have neither been integrated into the libraries nor into the applications. Enormous changes might take place making ECDSA more interesting.

This section will conclude with the comparison of realistic settings, i.e. examples for a different number of users, algorithms and key lengths. The generic formula of the computational as well as timely cost of a single round per user can be found in Equation 2. The appropriate values for the given algorithm, key length and number of users have to be specified. Table IV provides the run-times per prolongation of an idle powered scenario with different number of users and two pairs of key lengths for RSA and ECDSA of comparable strength.

$$|cost\_per\_prolongation\_and\_user| = |signatureCreation| \cdot outerRatio \cdot (1 + (n-1) \cdot innerRatio) \tag{2}$$

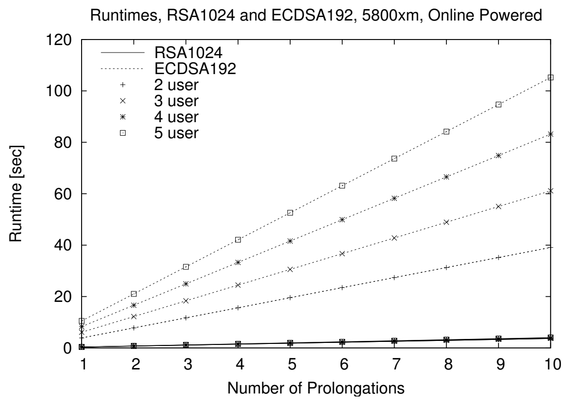
It became obvious that ECDSA is already in a two user scenario ways too slow, as a single prolongation needs about 4 seconds compared to 400 msec in case of RSA. The usage of ECDSA at these levels of security is not feasible. ECDSA is about a factor of ten slower than RSA in the optimal case with only two users, becoming worse the more users participate. RSA signature processing, even with a key length of 2048, which is considered secure for the next few years (until $\approx$ 2012), needs only a couple of seconds per prolongation. This analysis pointed out that multiple prolongations – mandatory for incremental contract signing – are possible by the use of RSA1024 within the short time period characteristic for ad-hoc scenarios.

TABLE IV

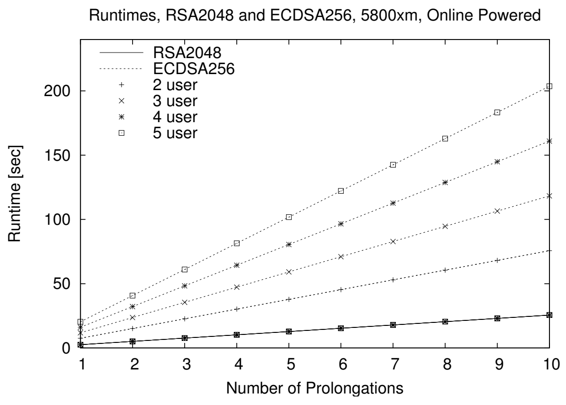
ESTIMATED TIME PER PROLONGATION, MULTIPLE USERS, ONLINE POWERED.

Algorithm & Key Length	Generic Formula	Number of Involved Users n=2	n=3	n=4	n=5
RSA1024	$(1+(n-1)\cdot 0.08)\cdot 7.2\cdot 50$msec	388 msec	417 msec	447 msec	475 msec
ECDSA192	$(1+(n-1)\cdot 1.30)\cdot 1.0\cdot 1750$msec	4025 msec	6300 msec	8575 msec	10850 msec
RSA2048	$(1+(n-1)\cdot 0.04)\cdot 53\cdot 50$msec	2756 msec	2862 msec	2968 msec	3074 msec
ECDSA256	$(1+(n-1)\cdot 1.30)\cdot 1.9\cdot 1750$msec	7647 msec	11970 msec	16293 msec	20615 msec

Figure 5 provides two plots for different key lengths in a setup with two to five users, depicting the timely effort needed for up to ten prolongations. It is a more detailed visualization than Table IV. The run-times of RSA1024 and RSA2048 are compared with the run-times of ECDSA192 and ECDSA256 as each part is of equivalent strength. ECDSA is represented by dotted lines, the continuous line in the figures is the run-time necessary if RSA is used. In contrast to ECDSA the different users can not be distinguished as their graphs do overlap. The situation is similar for both strengths, Figure 5(a) and 5(b).



(a) Run-times for RSA1024 and ECDSA192.



(b) Run-times for RSA2048 and ECDSA256.

Fig. 5. Run-times, Comparison of Two Equivalent RSA and ECDSA Key Lengths, 5800xm Online Powered.

ECDSA and RSA have been compared based on real measurements for different settings. It turned out that RSA is much more suitable to the idea of incremental contract signing due to the different costs of the mandatory operations, signature creation and verification. Besides this, the basic run-times are much smaller, allowing for the use on mobile devices with little computational power.

IV. Semi-Sequential ICS

The signing schema described before, named *parallelized signing*, is intuitive due to its complete parallelization of each prolongation: All users create a signature with the same expiration date at the same time during the first phase. Then, the necessarily sequential handover of the signatures will take place. During the last phase of each prolongation, the verification of all these signatures has to be done – all users in parallel, each of them one signature after another. As all users' signatures refer to the same expiration date, the order of the signatures' handover is not relevant. Figure 6 presents a variety of (handover) schedules. Common to all of them is, that signatures of the recent prolongation have to be exchanged, before any signature of the next prolongation will be handed over. Lets have a closer look at these handover schedules. The round robin *(rr)* one is quite intuitive, easy to set up and most likely be used, although the drawback is that during each round the same user has to go first. A completely chaotic *(cc)* approach can prevent this, without any drawback except setting up the schedule. This problem is also mitigated by the rotating rr *(rrr)*, where the user that goes first during each round also changes in a round robin manner. The bounce back *(bb)* schedule is related to *rr*, but reversing the order after each round. Now, the odd situation arises that the last user of each round has to hand over two signatures back to back. This phenomenon will be examined and exploited in the next section.

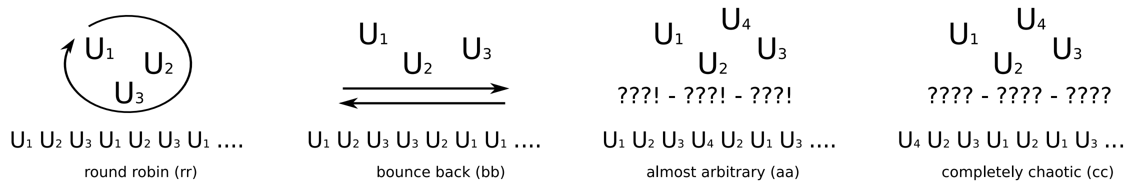


Fig. 6. Handover schedules: round robin, bounce back, almost arbitrary, completely chaotic.

A. Semi-Sequential Scheduling

An improvement by almost 50% is possible by adapting the handover and signing schema. So far, all users signed for the same expiration date and these n signatures are handed over and verified before the next round starts. The fact that the last user, handing over her signature for the current prolongation does not incur any risk at all – as she is in possession of all other users' signatures for the recent expiration date – can be exploited. A theoretical explanation and a formal model is given in [1]. Normally, each user has to contribute to incremental contract signing by incurring some small risk. To incur *some* risk at all, she has to hand over a signature lasting longer, most likely until the expiration date scheduled for the next prolongation. But then, this single signature can be used, instead of two subsequent signatures of the same user, for the recent and the next prolongation. One signature per two consecutive prolongations can be saved by this approach. The only prerequisite is, that this very signature will only be handed over after all other users' signatures for the recent prolongation have been received and successfully verified by this last user. This schedule is called *semi-sequential* because as many operations as possible are carried out in parallel, while the last user has a special role. She has to be known beforehand. The handover(s) as well as the verification of her signature can not be integrated in the parallelization that is possible among the other users.

We will address this fact later in more detail. All handover schedules except the *completely chaotic* one are suitable as the last user of each prolongation has to be chosen before signature creation starts. At this very moment, this particular user will be the only one creating a signature valid for the next prolongation's expiration date. The almost arbitrary *(aa)* handover schedule, where the last user of each round is determined before, is the least common denominator. The schedules can be found in Figure 7.

If this is exploited, one signature can be "shared" among two consecutive prolongations. Therefore, the number of signature processings is:

- gn are necessary in the parallelized setting, as n signatures during each of the g prolongations.
- $gn-(g-1) = g(n-1)+1$ are necessary in the semi-sequential scenario, as gn signatures are needed in parallelized one, but $g-1$ signature processings can be saved.

A closer look is necessary to rate the savings as the overall number of signatures depends on both, g and n, but the reduction depends only on g. Equation 3 describes the fraction of signatures compared to the parallelized setting for n users. Additionally the optimal case with $n=2$ is also contained:

$$\frac{gn-(g-1)}{ng} = \frac{g(n-1)+1}{ng} \qquad n=2 \Rightarrow \frac{g+1}{2g} \Rightarrow \approx 50\% \text{ for large } n \tag{3}$$

By the use of the semi-sequential schema the number of signature processing is reduced to $\frac{g+1}{2g}$. In the most likely case, where only two users are involved and the number of prolongations g is large, the achieved advantage is maximal with almost 50%. In case of a two user setting, every signature except the last signature of each user is dedicated to a different expiration date.

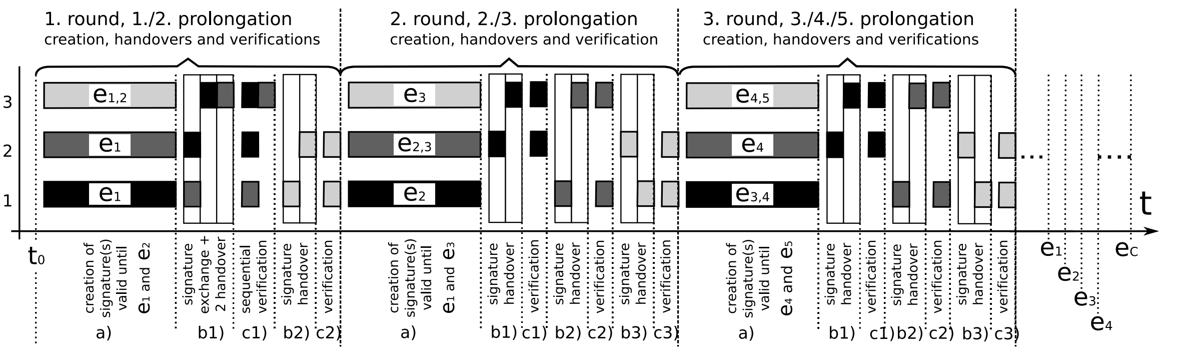

1. round, 1./2. prolongation
creation, handovers and verifications
2. round, 2./3. prolongation
creation, handovers and verification
3. round, 3./4./5. prolongation
creation, handovers and verifications
U3
U2
U1
e1,2
e1
e1
e3
e2,3
e2
e4,5
e4
e3,4
t
t0
creation of signature(s) valid until e1 and e2
signature exchange + 2 handover
sequential verification
signature handover
verification
creation of signature(s) valid until e1 and e3
creation of signature(s) valid until e4 and e5
a)
b1)
c1)
b2)
c2)
b3)
c3)
e1
e2
e3
e4
ec

Fig. 7. Semi-Sequential Schedule, 3 Users, 3 Rounds, 4 Prolongations.

A more complex setting with three users is depicted in Figure 7. Signatures are handed over in a round robin manner, which means that the last user of a prolongation is known before. Two subsequent prolongations overlap at this very user as her recent signature is shared among them. This happens multiple times in the figure: If a bar, indicating the signature creation, contains the label "$e_{i,j}$" this signature has to be used as the last signature to complete the signature set dedicated to e_i and as the first signature when assembling the set valid until e_j. But then, the signatures can not be exchanged any longer in a pairwise manner as these particular users would incur a higher risk. The signature exchange with this user has to be split into unidirectional handovers. Handover means, that the signature of this user are handed over to all other users, without immediately receiving their signatures in return. Please note that the number of signatures until a prolongation of the expiration date can take place has been reduced. Instead of n signatures only $n-1$ are needed for all except the first prolongation; the first prolongation still needs n signatures. Thus, the alignment between round and prolongation does not exist anymore, because of the reduced effort to complete a prolongation. More prolongations are possible with the same computational and timely effort. While in the *parallelized* scenario, shown in Figure 1, only three prolongations were possible within three rounds, the *semi-sequential* approach allows for four prolongations.

B. In-Detail Analysis

Please remind, that the savings discussed so far address only the overall number of saved signature processings. Both, the computational (battery power) as well as the timely effort respective savings can foster their exploitation. Until now, the discussion was focused on the number of signature processings, which means the computational cost each user has to shoulder. These saving can vary between users as prolongations are not necessarily aligned to rounds. Because of this some users may have to create one signature less, but they have to verify more signatures. The *inner ratio* is the mandatory information to calculate the distribution of the load among users during the last round. The computational effort is only distributed evenly, if the overall number of signatures is a multiple of the number of users, i.e. $gn-g+1=cn \leftrightarrow g-1=cn,\ c\in\mathbb{N}$. The computational savings are equally distributed is this case, as each user has to process c instead of g signatures.

The possible net gain in time by the use of semi-sequential signing depends strongly on the *inner ratio*. There are two relevant factors that are influenced by the *inner ratio*: The minor one is the unequal distribution of signature creation and verification, the not maximally exploited parallelization of the verifications is the major one. For the latter one, an additional time slot for verification has to be introduced to allow the other users to verify the last users signature. Instead of a single verification phase *c)* during the first prolongation, as shown in Figure 1, two verification phases *c1)* and *c2)* interleaved by the handover phase *b2)* of the last user's signature are necessary. Although the computational cost remains the same, see Equation 1, the timely cost increases by the cost of one verification:

$$|signature\_processing_{semiseq,\ time}| = |creation| + n\,|verification| \tag{4}$$

The impact of this additional effort can be easily estimated if the *inner ratio* is known. In case of RSA, with an *inner ratio* of about 0.08 the impact will be small regardless of a reasonable number of users. In contrast, the impact for ECDSA usage is tremendous as the *inner ratio* is about 1.3. Lets first examine some exemplary settings, shown in Figure 8 with 12 prolongations and a different number of users involved. Figure 8(a) shows the computational effort which is necessary for the semi-sequential schedule in relation to the parallelized one to achieve the given number of prolongations. In this plot, the cost of the parallelized setting is set for any number of user to 1. There are no savings possible in case of one prolongation. As shown before, the savings are maximal for two users and a large number of prolongations. The highest advantage is achieved if only two prolongation would take place.

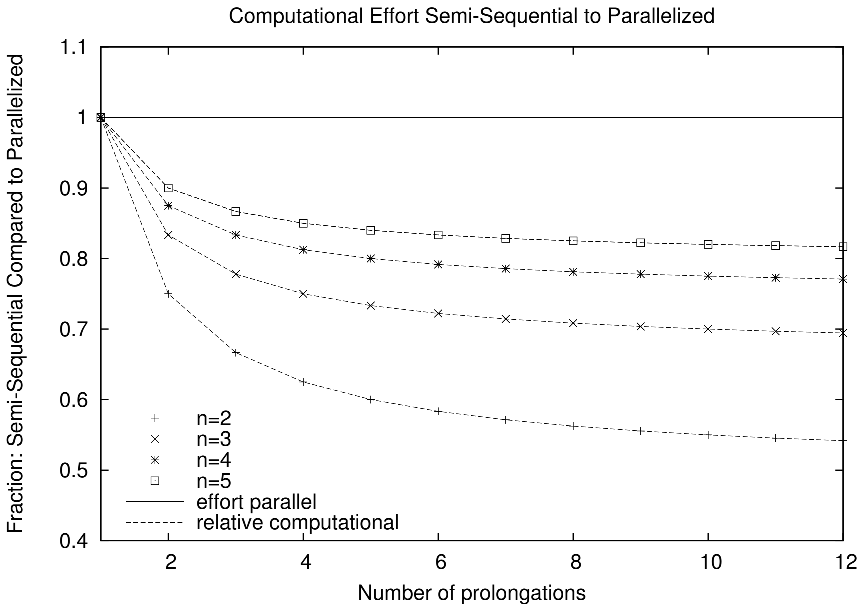
While the computational effort needed decreases continuously, the timely savings vary with a decreasing tendency due to the above mentioned whole-numbered ratio between the n users and the $g-1$ signatures saved. Plots of the timely savings are provided for both algorithms in a similar setup: RSA in Figure 8(b) and ECDSA in Figure 8(c). Although computational savings exists right from the second prolongation for any n, the timely savings begin at prolongation number $n+1$. The rate of the savings, if there are any, depends first of all on the *inner ratio* and, as already seen for the timely savings, on the number of users. The savings in case of RSA are significant, because an additional verification per prolongation is, due to the small *inner ratio (ir)*, of minor impact. Semi-sequential signing employed with ECDSA does results, due to its high *inner ratio*, only in minimal savings in a two user setting. If more than two users are involved, ECDSA is in the semi-sequential setting, due to its big *inner ratio*, much slower than the parallelized one.

Again, Figure 7 is a good example, because it depicts that only nine signatures are needed to achieve four prolongations in a three user setting. In the parallelized setting, $ng = 3 \cdot 4 = 12$ are needed, which means that the semi-sequential one achieves the same result with an effort of only 75%. The timely savings are in case of RSA 20%, for ECDSA about -3%.

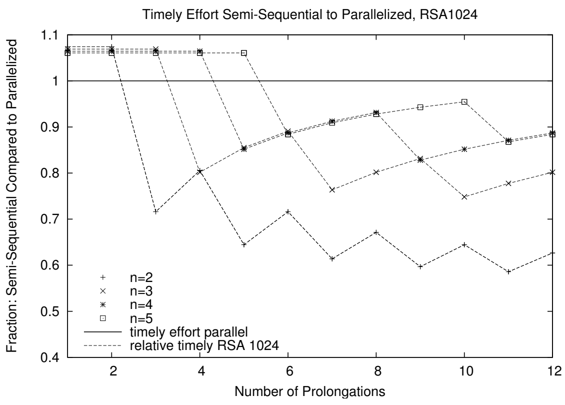
The *semi-sequential* signing schema is optimal regarding the number of signatures to be processed. Although it has only been briefly motivated, the result is equal to the formal findings when relying on the users' abilities derived from the reception of signed signatures from all others [1]. One drawback for semi-sequential signing has to be mentioned as it introduces more sequential operations: The signature exchange can not take place as a pairwise exchange if the prolongation's last user is involved because signatures dedicated to different expiration dates have to be exchanged. Closely related to this is the fact that the signatures can not be verified by all n users in parallel which also causes additional delays. Fortunately the two user scenario is not affected by these drawbacks. An evaluation of the impact resulting from the increased communication overhead will be provided in Section V and countermeasures will be presented. In contrast, the impact of sequential verification also depends on the algorithm that has been chosen and the number of users. Again, the *inner ratio* can be used to express the relative additional costs. Semi-sequential signing increases the number of sequential handovers and verifications by one per prolongation – similar to an additional user. Semi-sequential signing is recommended if the computational savings are of main importance. Regardless of the signing schedule, the usage of RSA with its small *inner ratio* is recommended to avoid any timing issues.

C. Summary

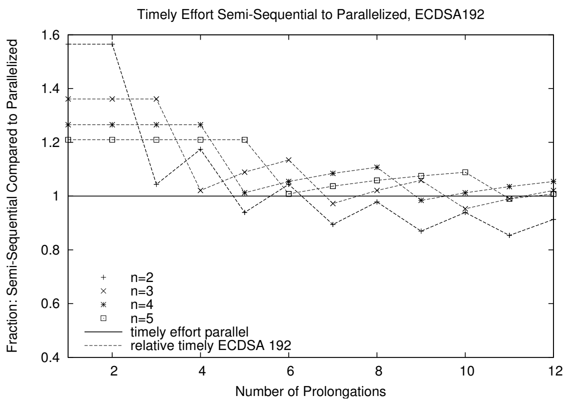
We were able to show that *incremental contract signing* is feasible by the use of widespread available public key algorithms on nowadays mobile devices. As neither an adaption of known algorithms nor any special purpose algorithms have been used, any digital signature algorithm can be used. RSA and ECDSA have been chosen, based on theoretical considerations presented in Section VI-F, to be evaluated for the creation and verification of signatures. It turned out that



(a) Computational Effort.



(b) Timely Effort, RSA, ir=0.08.



(c) Timely Effort, ECDSA, ir=1.3.

Fig. 8. Comparison: Parallel and Sequential Scheduling, Timely and Computational Cost.

ECDSA is quite slow despite its short keys and does not scale well with an increasing number of users. Unfortunately, only a single cryptographic library was suitable to our policies, so that no comparison between different implementations was possible. But, significant differences in the ratio of RSA and ECDSA have been discovered among the devices. Therefore a continuous evaluation of the software development and hardware releases is necessary. The operations that have to be carried out just-in-time are possible during an ad-hoc meeting where a reasonable number of prolongations is demanded if RSA is employed. This holds for both signing schemes, the naive, parallelized schedule and the semi-sequential one. By the use of RSA in the semi-sequential signing schema an improvement of almost 50% can be achieved in the most likely two user setting. Thus, strong cryptography is possible, which enables the novel approach of incremental contract signing on nowadays mobile devices.

V. Communication

We envision that applications built on top of our libraries provide a responsive, well featured graphical interface to the user and handle the entire communication with other devices in the background – besides all other tasks. Still, the emphasis is on Bluetooth communication as it is widespread available on cell phones although certain drawbacks do exist. As part of the digital ownership research project, a Bluetooth communication library, *BtCom*, has been developed to suffice these needs and enable communication without bothering about details. Therefore the application programmer can rely on a high-level API providing message-based communication. Although we do focus entirely on the communication ability, much more functionality such as improved device-discovery and service-inquiry has been integrated. BtCom has already been used in other libraries providing FTP-like access of the mobile device's persistent storage or http-request tunneling over Bluetooth to save air-time as well as applications, e.g. a mobile trading card game. The following measurements estimate the round trip time *(RTT)* from the API method invoked to send the message until it is received by the receive method of the originating device. The message is returned by the responding device at the "application's" application level, not somewhere down in the libraries. The message has to pass the API twice on each device, approximating a real answer/reply scenario. Additional, the message is not returned, but another one with different content is assembled and sent back.

Both, a responsive user interface and communication have to be designed carefully with respect to the available resources. Even more, if they have to be used together. All network issues and message handling as well as user interface drawing and interaction processing have to be done in separate threads. Especially the intervals for buffer and queue examination have to be chosen carefully. Depending on the desired responsivity different polling intervals can be employed for all of the mentioned areas. Although this are all interesting aspects of programming mobile devices, the communication is our main concern: While a communication delay of more than 1000 msec for a chat application is tolerable, the round trip times for interactive applications such as games have to be much smaller. Even tighter restrictions can be used if no interaction between users, but simple data exchange takes place between devices. We are not going into the difference between a general purpose library and an optimized communication setup, which is able to achieve round-trip times of about 170 msec. Table V shows the round-trip-times for a message containing an one kilobyte large payload. The measurements provide the values that have been derived for minimum, median and average from the exchange of a few thousand messages. A payload of 1024 bytes has been assembled from 64 chunks, each 16 bytes long. It turned out that the periods of higher RTTs observed on some devices resulted from the user interface waiting for input. As soon as the screen saver became active, the RTTs dropped. Unfortunately, we also observed the different behavior in form of a better performance as long as the device was waiting for user input. Although these events (screen saver activation and deactivation) will not happen in a normal scenario, the knowledge about their impact on the RTTs is of importance. It might be reasonable to forbid or even deactivate user interaction to speed an ongoing combined communication and signing process up. The table contains measurements for pairs of devices, where the device specified by the row sends, the device noted in the column replies, i.e. receives the incoming message and sends its own in return. Because of this, measurements between disjunct devices are not symmetrical. 3000 messages were send for each device, which resulted in 3000 replies. It turned out that the RTT for a message with 1kb payload, which would be sufficient for signature exchange, is far below 1 sec.

TABLE V
RTT – AVERAGE RTTS. PAYLOAD SIZE 1KBYTE. MINIMUM / MEDIAN / AVERAGE[MSEC].

Devices	M600i	5800xm	6300	E50	K800i
M600i	481 / 512 / 525	403 / 479 / 471	442 / 495 / 503	386 / 479 / 479	506 / 714 / 723
5800xm	327 / 369 / 377	–	186 / 325 / 316	235 / 623 / 467	298 / 392 / 448
6300	451 / 487 / 515	221 / 368 / 330	248 / 390 / 407	340 / 377 / 378	469 / 556 / 608
E50	526 / 643 / 645	616 / 689 / 692	559 / 690 / 668	–	668 / 706 / 753
K800i	491 / 635 / 656	268 / 366 / 386	407 / 528 / 551	235 / 454 / 417	513 / 741 / 750

A different kind of test series have taken place to get an idea of the overhead resulting from the assembly of the payload and the bits transfered per second. In this setting only the three pairs of identical devices, 6300, K800i, M600i, were used. The measurements for different payloads, each consisting of 128 chunks, are provided in Table VI. Again, 3000 messages have been send in each direction. In addition, a 0kb payload has been used, i.e. an empty packet to learn about the overhead introduced by the layered architecture of the library and application and the air interface. It turned out, that the majority of the round trip time resulted from handling the packet. The difference between RTTs of different payload sizes is small compared to the RTT of an empty packet. The bit rate per second, which increases with the increasing payload is the indicator. The larger the payload is, the more efficient the communication is. All devices support Bluetooth 2.0, the 6300 and the K800i also EDR.

TABLE VI
RTT (MIN / MED / AVG) [MSEC] AND BANDWIDTH [KBPS] BETWEEN PAIRWISE IDENTICAL DEVICES.

Devices	0kb		1kb		2kb		4kb	
6300	236 / 371 / 386	–	250 / 391 / 410	21	258 / 400 / 428	41	260 / 400 / 432	82
k800i	384 / 479 / 492	–	513 / 875 / 874	9	738 / 1067 / 1083	15	926 / 1460 / 1476	23
m600i	339 / 383 / 390	–	477 / 510 / 533	16	710 / 777 / 813	21	873 / 1029 / 1046	32

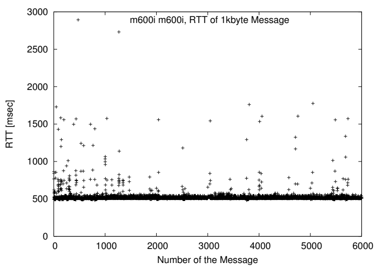
A. Observations

During our test runs we were able to observe the influence of (waiting for) user interaction in contrast to an active but inexpensive screen saver. Additional to this we discovered quite a variety of different RTT pattern. For your tests, bursts of 1000 messages were sent, i.e. the user was interacting with the device at least every 1000 messages to start the transmission. Therefore the screen saver was not active every 1000 messages for a certain amount of RTTs. During the test runs, the screen saver inclusive keyboard lock was allowed to become active to learn about its impact. Additional user interaction took place to verify the impact of interrupting the screen saver and forcing the reactivation of the user interface. We present the RTT measurements by the use of two plots for each setup: First, the RTTs are plotted along the timeline and then as a cumulative function. Figure 9 shows the plots for the three setups where two identical devices exchanged multiple thousands of messages. The first half of the measurements results from one device, the second half from the other device. Messages were alternatingly exchanged in bursts of 1000 messages to learn about the a possible influence of the device on its own performance. The plots of Figure 10 contain measurements, again in form of plain RTTs and the cumulative function, for a single burst between two devices. Figure 9(a) shows the communication between two Sony Ericsson M600i with the majority of the RTTs at about 500 msec and few longer RTTs. Neither an influence of the screen saver and the automatic key lock nor a difference between the two devices could be detected. In case of the communication between two K800i, shown in Figure 9(b), two accumulations can be found at 600 msec and 750 msec – although the RTTs are scattered up to 1100 msec. The difference between screen saver active and inactive can not be revealed in this plot, but had been identified in close ups as little black spots at 620 msec and 690 msec but without scattering up to 1100 msec. This is somehow strange as the two K800is seem to communicate faster and with more constant RTTs if the screen saver (black screen with simple digital clock) is not active. Figure 9(c) provides the RTTs between two 6300, which are very precisely accumulated around two, almost three RTTs, 270, 400 and 500 msec. The influence of the user interactions can be seen by short black spots which result from an RTT increase of about 50 msec if the user interface is active. A slight difference between the two devices can be as the first 3000 messages have been sent from one device, the last 3000 messages from the second 6300. We do not have an explanation for this minor bias as both devices were identically configured and restarted before each test run. A single burst of 1000 messages between these devices can be found in Figure 10(a). Figure 10(b) and 10(c) show the communication measurements between two different devices known from the plots before, one K800i and one 6300. As the K800i and 6300 have quite individual RTT patterns the RTTs between different devices can not be clearly classified. It can not be distinguished between delays resulting from the initially sending or the responding side. But, the scattered RTTs result from the sending K800i, not from the responding K800i. The influence of the deactivated screen saver, is much more obvious than in the plots provided before.

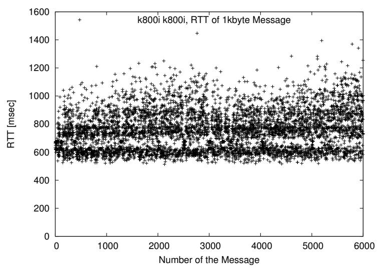
Because 1000 resp. 6000 measurements are plotted in each graph addressed so far, the cumulative distribution is also provided to visualize the quantity of measurements in each accumulation. These accumulations cause "steps" in the cumulative function. It can also be seen that for all settings without an K800i being involved, very precise RTTs for the majority of packages sent can be stated. But, the individual device has to be considered as the screen saver can have positive, none or even negative impact on the performance.

B. Complexity of Communication

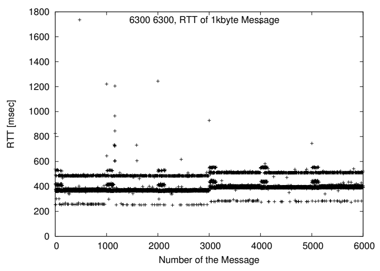
The measurements estimate the communication necessary for the signature exchange between two users, i.e. their consecutive handover of one signature each. It turned out that the RTTs do not differ significantly for the different examined payloads: 0, 1, 2 and 4kb. Therefore, it is almost arbitrary if a small acknowledgement or a big chunk of data is sent. This simplifies the analysis because the measured RTTs can be used for both, pairwise exchange of signatures and the unidirectional handover. These findings will be applied to the parallelized and the semi-sequential scenario in the



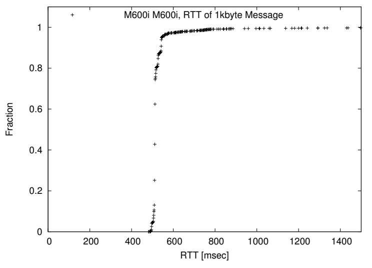
(a) RTTs, Two M600i.



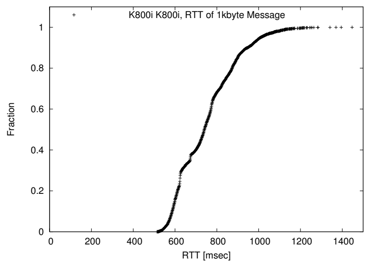
(b) RTTs, Two K800i.



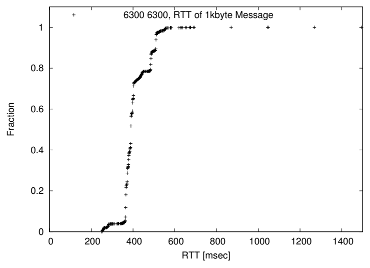
(c) RTTs, Two 6300.



(d) Cumulative RTTs, Two M600i.

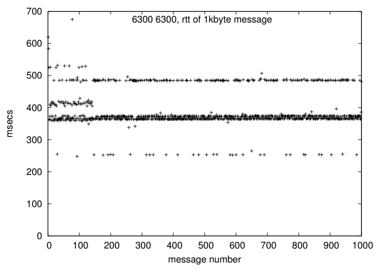


(e) Cumulative RTTs, Two K800i.

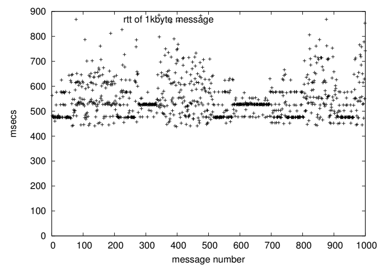


(f) Cumulative RTTs, Two 6300.

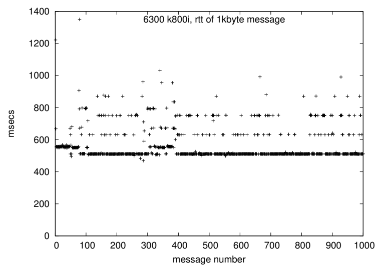
Fig. 9. Round Trip Times and Cumulative Function of Symmetric Device Constellations, 3 Bursts each, interleaved.



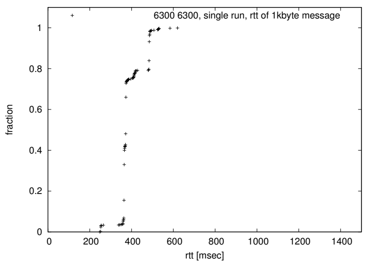
(a) RTTs, Two 6300.



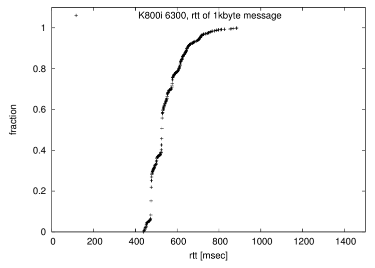
(b) RTTs, K800i and 6300.



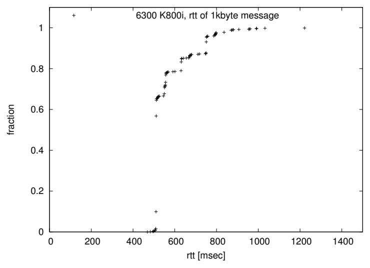
(c) RTTs, 6300 and K800i.



(d) Cumulative RTTs, Two 6300.



(e) Cumulative RTTs, K800i and 6300.



(f) Cumulative RTTs, 6300 and K800i.

Fig. 10. Round Trip Times and Cumulative Function, Single Burst.

following sections. Furthermore, the suitability of piggybacking signatures will be introduced and the possible gain be discussed.

1) Parallelized Signing:

In the *parallelized* setting, where pairwise signature exchanges between n users are possible, only $\frac{n(n-1)}{2}$ communications have to take place per prolongation. Such a bidirectional signature exchange, where no verification of the received signatures takes place before handing over the own signature, is harmless, as all signatures are dedicated to the same expiration date. Besides this most efficient communication pattern, the communication and the verification operations can be clearly separated in the two phases: signature exchange *[b]* and signature verification *[c]* as shown in Figure 1. Only then, the entire parallization of verification is possible which can result in significant time savings.

- Therefore, the overall number of communications is
  - $g\frac{n(n-1)}{2}$ if n users are involved and g prolongations take place.

The complexity is in $O(n^2)$ per prolongation.

2) Semi-Sequential Signing:

In the *semi-sequential* setting, communication and verification can not be clearly separated in two phases and carried out without interruption. The single operations have to be interleaved, because the last user of each prolongation hands over a longer lasting signature, used within two consecutive prolongations, *after having received and verified all signatures of the recent prolongation*. Because a pairwise exchange is not possible with the last user of the recent prolongation, which is also the first user of the next prolongation, additional handovers have to take place in a setting with more than two users. The communication and verification have to be interleaved as shown in Figure 7. Unfortunately the pairwise handover (signature exchange) between a subset of the users is only possible if four or more users are involved.

- The communication effort during the first prolongation is:
  - $\frac{(n-1)(n-2)}{2}$ pairwise handover(s) between the first $n-1$ user(s),
  - $n-1$ handover(s) towards the last user and
  - $n-1$ handover(s) from the last user to the first $n-1$ user(s).

    $\Rightarrow \frac{n(n+1)}{2}-1$ altogether for the first prolongation.
- The communication effort for every subsequent prolongation is:
  - $\frac{(n-2)(n-3)}{2}$ pairwise handovers between all except first and last user
  - $n-2$ handover(s) towards the last user and
  - $n-1$ handover(s) from the last user to the first $n-1$ user(s)
  - $n-2$ handover towards the first user.

    $\Rightarrow \frac{n(n+1)}{2}-2$ altogether for every subsequent prolongation.
- Therefore, the overall number of communications is

  $\Rightarrow g\frac{n(n+1)-4}{2}+1$.

The complexity is in $O(n^2)$ per prolongation.

3) *Piggybacking:*

Until now, the communication patterns have been derived from the needs of the cryptographic operations. Another approach is to piggyback the signatures, but still adapting the pattern imposed by the ccryptographic operations. Employing communication in a round-robin manner, i.e. only n pairs of users need to communicate by forming a circle. Then, the complexity of communication during a single prolongation is $O(n)$ instead of $O(n^2)$.

Although the size of the payload per communication is now n-times higher, it is much more effective because of the higher bandwidth that can be achieved. As shown before, transmitting more bytes per transmission is cheaper than multiple transmission with less payload. The communciation speedup by piggybacking can be applied to both cryptographic schedules, the parallelized and the semi-sequential one. Although the cost of communication is reduced, the gain can be reduced or even negated by additional (timely) cost for the sequential verifications. By the use of piggybacking only signature creation can be parallelized. Therefore a thoroughly evaluation of the single cost factors, communication and signature verification have to take place. Then, the timely cost for every prolongation except the first one is

$$|time\_per\_prolongation_{piggybacked}| = |creation| + (n-1)(n-1)|verification|. \tag{5}$$

C. *Comparison*

So far, two communication patterns adapted directly from the cryptographic plot have been presented. The principle of piggybacking signatures to reduce the number of communications has been introduced and can be applied to both signing schedules. These two will also be respected during the following comparison. The drawback of introducing more sequential verifications is crucial, i.e. it has to be considered when rating the gain achieved by signature piggypacking. A comparison of the communication effort of these four approaches in typical setups with ten prolongations and a different number of users, $n = 2,\dots 5$ is presented in Table VII. The costs for a single communication between the different device combinations have been determined before; a round trip time of $rtt = 600$ *msec* is assumed. About 80% of the measurements, as shown in the Figures 9 and 10, meet this limit.

TABLE VII
NUMBER OF COMMUNICATION AND TIME NEEDED IN DIFFERENT SETTINGS.

communication scenario	formula	amount of communications and time needed							
		n=2		n=3		n=4		n=5	
adapted parallelized	$g\frac{n(n-1)}{2}$	10	6.0 sec	30	18.0 sec	60	36.0 sec	100	60.0 sec
adapted semi-sequential	$g\frac{n(n+1)-4}{2}+1$	11	6.6 sec	41	24.6 sec	81	48.6 sec	131	79.0 sec
piggybacked parallelized	gn	20	12.0 sec	30	18.0 sec	40	24.0 sec	50	30.0 sec
piggybacked semi-sequential	$g(n-1)+1$	11	6.6 sec	21	12.6 sec	31	18.6 sec	41	24.6 sec

The communication for both native schedules, the parallelized and the semi-sequential, does not scale with respect to the users involved as their complexity is in $O(n^2)$. Fortunately this is not an issue for the most likely two user scenario. Piggybacking does not make sense in a two user scenario, a simple pairwise exchange of signatures is more effective. Nevertheless, the theoretical costs are also listed in Table VII. The two user parallelized setting is as fast as possible, the semi-sequential one is with an increment of $1/g$ slightly slower. Therefore, the decision which schedule to use has to be based on the timely difference resulting from the cryptography. But, the cost for communication differs dramatically, if more than two users are involved. Although both piggybacked communication patterns are faster, the timely impact of the sequential verification has to be considered. Please remind that the proposed optimizations assume that verification can be done in serial instead of being parallelized due to its short runtime compared to transmission. This ratio between creation and verification of a signature is expressed by the *inner ratio*. The smaller the *inner ratio* is, the more suitable is this algorithm and key length combination for sequential verification. The proposed optimizations are therefore not suitable for all cryptographic algorithms, because these result in a trade-off between communication and verification speed as verification has to be done in serial instead of parallelized by all users.

VI. Platform, Infrastructure and Algorithms

Because the vision of Digital Ownership is relevant to almost everyone, we decided to target only widespread available mobile platforms. These have to be able to store and playback reasonable amounts of digital content and provide sufficient computational resources to allow for strong cryptography. The access to short range communication is mandatory to get into contact with other devices. Infrastructure based data-communication such as GPRS is of advantage, but optional. We have chosen mobile phones as most PDAs were targeted at the business customers, too expensive and not widespread used. There was also a lack of the mandatory short range communication abilities as well as accessible APIs. Another important factor was the platform and programming language our libraries and applications should use. Sun's Java Platform, Micro Edition, short Java ME, [9], has been chosen. First of all it seemed to be easy enough to use and that some source code could also be reused for the server side as Java ME contains a subset of Java SE's functionality. While this worked out for the first evaluation runs, the server side has been migrated to Ruby on Rails [10] as it became clear that its is more suitable for further prototyping and later usage in a productive environment.

This section is structured as follows: First we will introduce the Java ME platform, then an overview of Java's Cryptography Architecture will be provided, followed by a closer examination of potential Java Security Provider. The next two subsections are dedicated to digital signatures including hash functions and an in detailed examination of a choice of digital signature algorithms. The last subsection contains a conclusion and recommendations on the keylengths that should be used.

A. Introducing the Java Platform Micro Edition

The *Java Platform, Micro Edition* (Java ME) from *Sun Microsystems* [11] is a derivate of the *Java Platform, Standard Edition* (Java SE) with focus on the special demands of limited devices. Like

the Java SE Bytecode, every program compiled in Java ME Bytecode can be interpreted by a Java ME virtual machine, called KVM (K for kilobyte) instead of Java SE's JVM. Since mobile devices are less powerful than desktop computers, they are referred as limited devices, meaning their CPU is rather slow and their memory is small compared to machines supporting the Java SE. The Java ME consists of three parts: *configurations*, *profiles* and *optional APIs*.
The ***configuration*** is the base part, determining a subset of Java SE's API. It depends on the hardware capabilities of a device which configuration is supported. Two different configurations exist: The *Connected Device Configuration* (CDC) used on faster PDAs, a minority of smart-phones or set-top boxes and the *Connected, Limited Device Configuration* (CLDC), implemented often in cell phones. Because of the wide spreading of Java ME compatible cell phones, our cryptography libraries have to run on the CLDC as the lowest common denominator.
Profiles expand the API with device-specific user interfaces and storage functions; cell phones are usually compatible with the *Mobile Information Device Profile* (MIDP), which will be used throughout this paper, and PDAs use the *Personal Digital Assistant Profile* (PDAP). Although CDC profiles are more flexible and offer a more complex API, they will not be target of our development due to the self-imposed restrictions, i.e. too less devices employed.
Optional APIs may be used on top of the profiles, like a Bluetooth or 3D gaming API for a specific MIDP compatible device. Figure 11 provides an overview, please refer to [12] to get a deeper understanding of Java ME.

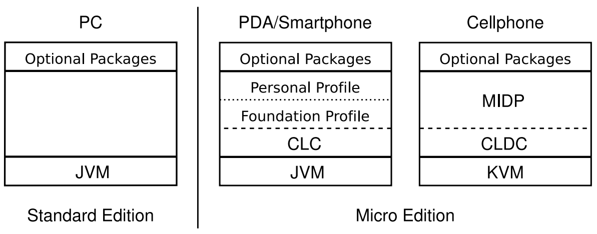


Fig. 11. Java SE and Java ME Stack Components

The Java ME's CLDC configuration with MIDP profile provides only a small subset of the bigger Java SE, which affects complete parts of the Java language a high level language programmer is accustomed to. For example, the CLDC 1.0 does not support floating point numbers at all, while they are implemented in CLDC 1.1, which makes the newer version recommendable.
Unfortunately, still the complete API for cryptographic computations is absent in these versions. Meanwhile, MIDP 2.1 is available on a small number of phones while MIDP 3.0 has been announced in February, 2009. To allow usage of cryptographic libraries written in Java, the mobile phone has to support CLDC 1.1 and MIDP 2.0 as minimal denominator and target platform for our (cryptographic) libraries and further development. Another requirement mentioned in the beginning was that the devices provide access to their short range communication abilities, i.e. Bluetooth. Although many devices have a Bluetooth interface to synchronize with a PC or to connect to a wireless headset, this is not sufficient. To enable Java ME to access the communication device JSR-82 has to be supported. Besides all this, additional pitfalls caused by incorrect and incomplete implementations have to be managed.

B. Java Cryptography Architecture in Java SE and ME

To understand Sun's two different cryptographic modules for the Java language, a brief overview at history is helpful. From version 1.1, Java is bundled with the module *Java Cryptography Architecture* (JCA), a library with interfaces for signing and hashing. In Java 1.2, another API was introduced: the *Java Cryptography Extension* (JCE), offering functions for several symmetric and asymmetric algorithms. But due to U.S. export regulations in force at that time, cryptographic source code was treated the same way as weapons and the export of the *Java Software Kit* to other countries was prohibited. Therefore, Sun split the JCA and the JCE and put all classes affected by U.S. export laws into the JCE and the non-prohibited into the JCA, enabling delivering Java with the JCA outside the U.S. - without the JCE. The laws changed in 1999, making it possible to bundle Java version 1.4 with a weaker JCE for export. Weak in this context refers to the maximum key length of the supported algorithms, allowing to easily break encryption by brute force. Today, after another change of the regulations in 2004, the only difference between the weak and strong JCE is a file containing security policies. U.S. law permits downloading and replacing the file with one from Sun's website to gain "unlimited strength" support.
The JCA offers a set of classes in the package `java.security` for transparent implementation and use of cryptographic functions like encryption or hashing, which enables other developers than Sun to design own cryptographic service providers. The programmer does not need to worry which certain *cryptographic service provider* is used. In Java ME, Sun excluded the JCA from the CLDC to downsize the consumption of disk space on mobile devices, but some providers created special mobile editions of their frameworks that also regard the technical circumstances on mobile devices.

C. Java Security Provider

Several security providers exist which enhance Java's cryptographic abilities. Only the more widespread are discussed here to benefit from a bigger number of testers, which guarantees greater code maturity. The criteria to evaluate the security providers are pretty much the same as for the digital signature algorithms in Section VI-F: Besides the availability of RSA and ECDSA, all implementations of discussed algorithms should be flawless regarding security aspects, the sources should be open (also to review the implementation) and free of charge. Ideally, a separate mobile edition exists that reimplements Java ME's lack of necessary data types for cryptography like `BigInteger` and is optimized regarding performance on mobile devices.

1) Sun JCE: Sun's JCE is the standard *cryptographic service provider* in Java SE and can be found in the package *javax.crypto*. It supports a variety of symmetric algorithms, but only a few asymmetric like RSA, while ElGamal, DSA and ECDSA are not implemented. The source code is not available and it cannot be integrated into mobile projects because of lacking classes like `java.math.BigInteger` and `java.math.SecureRandom`.

2) SATSA: The *Security and Trust Services API for J2ME* (SATSA) by Sun realizes parts of the JCE as optional API adding cryptographic functionality to Java ME [13]. It implements the digital signature algorithms RSA and DSA, but no elliptic curve cryptography. By supporting *smart cards* as security elements, cryptographic operations can be performed in a trusted environment on mobile devices. Unfortunately, devices are not necessarily supplied with smart cards (or the programmer does not has access to them, like a SIM card in a cell phone), nor any cryptographic hardware, so the functionalities have to be implemented in software, since the cryptographic provider has to be runnable on every Java supporting mobile device. Although SATSA is free to use, its source code is not available.

3) Bouncy Castle: The *Bouncy Castle Crypto API* [8], developed by the *Legion of the Bouncy Castle*, is a completely free and open source cryptographic service provider. It is a *Clean-Room-Implementation* of the JCE (meaning, that it supports the same interfaces, but is developed from scratch) and supports every algorithm asked for - and even a lot more. A *lightweight version* for limited mobile devices is available, too, which realizes the missing `BigInteger` and `SecureRandom` for use with Java ME.
Bouncy Castle is a voluntary project. In the beginning, the lack of documentation, examples and support resulted in a challenge to get it to work. A good point to start is [14]. While Java SE offers the well-documented JCA interfaces for use with Bouncy Castle, the lightweight ME version requires direct work with the Bouncy Castle classes, involving examining the source code for understanding the gearing of the package.

4) Cryptix: The *Cryptix* project [15] is another implementation of the JCE. Although it supports many algorithms, it lacks support for elliptic curves and offers no Java ME suitable version. PDA support is available and has been used for example in the *iClouds* of the *Darmstadt University of Technology*, [16]. Since support and development on the project seem to have stopped in 2005, it cannot be considered an option for our cryptographic applications. A subproject for elliptic curves, *Elliptix*, is in a pre-alpha state since 1999 and has not been updated anymore. Both projects came from a spin off from Systemics in the year 1999, see [17] for more details.

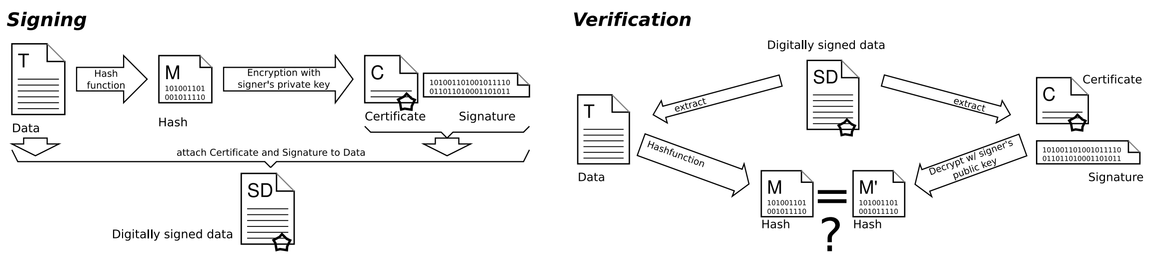
5) IAIK: The *Institute for Applied Information Processing and Communication* (IAIK) of the *Graz University of Technology* [18] provides a implementation of the JCE that is free of charge for research and educational purpose, but not for commercial use. A separate lightweight mobile version, the *IAIK JCE-ME*, is available. The IAIK JCE has a lot of different security algorithm schemes implemented and an optional API for elliptic curve cryptography is offered. The main advantage over Bouncy Castle is the availability of support for IAIK's products, the drawback of the commercially orientation are its costs and the fact of closed source. IAIK advertises the speed of the algorithms as a design focus.

Conclusion: It is quite disappointing, that besides build-in functionality such as *https* no support for generic cryptography does exist from the manufacturers. Due to the necessity of additional hardware, such as specific SIM-Cards, SATSA is not feasible for widespread and inexpensive use. Because of our policy to foster usage of open-source and freely available software, the use of the IAIK libraries was not the first choice. As the development of Cryptix, one of the two open-source projects, stopped in 2005, only Bouncy Castle was left for evaluation. The Bouncy Castle project is an open-source project under continuous development and both, voluntary reports on the internet and project documentation have improved over the last few years. All algorithms and platform targeted by our project are supported and well-functional.

D. Digital Signatures

To compensate the slowness of public key algorithms, it is not recommendable to sign the whole document t, but a hashed version of the document $m = h(t)$. This also solves security concerns of some algorithms like *existential forgery* in RSA [19]. Hash functions are covered in Section VI-E.

When M is the *message space* and C the *cipher text space*, signing a document $m \in M$ means to perform the *encryption* function $E : M \to C$ on the document with the private key of a public key algorithm and to obtain the signature $c = E(m)$, with $c \in C$. Only the owner of a key pair knows the private key and without it, nobody else can sign the document in the same way.
The signature can be verified by taking the signature and a public key as arguments and compute the *decryption* function $D : C \to M$ to get the encrypted message $m' = E(c)$, with $m' \in M$. If the private and public key belong together, the document itself will be the result of the operation, so that $m = m'$ (see Figure 12).



(a) Creation of Digital Signatures. (b) Verification of Digital Signatures.

Fig. 12. Creation and Verification of Digital Signatures

Key pairs must not be used additionally to encrypt or sign data besides authentication, because this is of high risk to security. If *Eve* pretends that she wants to prove *Alice's* identity, *Eve* is supposed to send a random number to *Alice* as challenge. In fact, the random number is the hashed value $h(m)$ of the text m. Alice cannot recognize this and signs the hash value. Actually, when sending the signed hash to *Eve*, *Eve* receives a self-chosen document signed by *Alice* [19]. A solution to this is the use of different hash functions for authentication and encryption, which produce different hash value lengths, so that the length of a hash value determines its purpose.

E. Hash Functions

A message digest h is a preferably random fixed-length representation of a message m of arbitrary length, created by a one-way hash function $h = H(m)$. One-way hash functions are hash functions with additional characteristics. While it is easy to compute h from m, the other way finding m from h is hard. Also, it is hard to find another message m' to m, such that $H(m) = H(m')$. Message Digests are used by digital signature protocols to shrink the number of bytes to sign, because signing is slow and message digests, ranging from 160 bits to 512 bits, are usually shorter than the original message. Several signature protocols even need hashing for security reasons (see DSA in VI-F.4).

Reducing a message to a shorter representation, comparable to a fingerprint of the message, may enable Eve to find two random messages m and m' with $H(m) = H(m')$, also called *birthday attack*. If it is hard detecting two *random* messages with the same hash value, the algorithm is called *collision-resistant*. Using a non-collision-resistant algorithm for digital signatures alleviates the search of different messages with the same hash value, so that it cannot be determined, which message was intentionally signed.

A popular family of hash functions, the MD (short for *Message Digest*) family, has been designed by Ronald Rivest, with the fifth one, MD5 from 1991, being the newest and strongest. While it is mainly used to create checksums of files available for download or storing passwords in databases, it is considered insecure due to its hash length of only 128 bits, allowing birthday attacks. In 2005, Lenstra, Wang and de Weger showed two different certificates with the same hash sum [20].

Another family of hash functions is the SHA family (short for *Secure Hash Algorithm*), proposed by the NIST as a *Federal Information Processing Standard (FIPS)*, see [21], and being technically similar to the MD family. The most common representative is SHA-1, which is often used in a variety of cryptographic applications. Its predecessor, SHA-0, has already been proved to be vulnerable against collision attacks, reducing its complexity from 2^{80} to 2^{39}, so it can be considered broken [22], while SHA-1's complexity has been reduced from 2^{80} to 2^{63} [23]. This makes SHA-1 not insecure yet in signing applications, because finding a document with the same hash value as another already signed document is still not feasible, but better attacks are expected to come. The family was enlarged by the SHA-2 algorithms SHA-224, SHA-256, SHA-384 and SHA-512, each creating according hash lengths, while the variants SHA-0 and SHA-1 create 160 bits hashes. The SHA-2 group is technically very similar to SHA-1: Although no weaknesses of SHA-2 have been found yet, they are expected to come. Nevertheless, the longer hash sums provide secure use for the next years, as long as no new critical mathematic weaknesses in the SHA family are discovered. All SHA-2 algorithms are covered by a U.S. patent held by the NSA, while SHA-1 is free to use.

A third group of hash functions is the *RIPEMD* (*RACE Integrity Primitives Evaluation Message Digest*) family. The first member of the group, the original RIPEMD, is considered insecure due to collisions found for MD4, on which RIPEMD is based [24]. Therefore, RIPEMD-128 was developed, producing a 128 bits hash length like RIPEMD, but seeming collision resistant yet. A stronger variant is RIPEMD-160, computing 160 bits output, while two other successors, RIPEMD-256 and RIPEMD-320, only reduce the risk of collisions. The algorithms are developed in an open community and none of them is covered by patents [25]. Compared to SHA-1, RIPEMD-160 is faster, but not as widespread and well-investigated as NIST's algorithm. Since security is more important than speed, use of SHA-1 is recommended.

F. Choice of Digital Signature Algorithms

A public key algorithm that is able to work as a digital signature algorithm has to be chosen considering our demands of cryptography on mobile devices – encryption and signing. Only three public key algorithms are capable of signing besides encryption: RSA, Rabin and ElGamal [26]. A fourth one, the *Digital Signature Algorithm* (DSA), cannot be used to encrypt. These algorithms are discussed in this Section. A special variant of the DSA, the *Elliptic Curve DSA* (ECDSA), is described in an own subsection. The criteria are, in order of importance:
Security. Ideally, keys will be valid and not compromised for years, representing an union with the associated digital identity[3]. Security aspects of the algorithms are possible vulnerabilities of the underlying mathematical techniques and appropriate key lengths to avoid breaking key pairs via brute force in an estimated period of time.
Availability. The algorithm should be freely available and the implementation open source to allow for thoroughly examination. Although software patents are not applicable in the European Union, an algorithm should be free to use worldwide.
Speed. With each new mobile device generation, processing units become faster, but mobile devices still are rather slow compared to desktop computers. Cryptographic calculations demand operations on very large numbers, while specialized coprocessors are usually not available. The level of security needed for the desired operations, e.g. a key pair used for authentication for several years versus one time contract signing valid for a couple of days. The stronger the cryptography, the longer operations last. Because of this the strength, i.e. the employed algorithm and key length has to be chosen carefully.
Memory Consumption. Mobile devices have very limited amount of memory, which allows only for applications with a small footprint. In contrast, build-in persistent storage has a capacity of a few dozen of megabytes, whereas exchangeable memory cards are capable of several gigabytes. Persistent storage becomes important at the latest if digital content has to be carried around.

1) RSA: RSA is an abbreviation for the three inventors of this algorithm: Rivest, Shamir and Adleman. It was the first public key cryptography algorithm developed and it is the most frequently used today. Encryption and signing are possible and it is easy to implement. Due to its popularity, RSA is well-investigated and countermeasures can be taken to all possible attacks discovered yet (e.g. *Chosen Ciphertext Attacks* or *Low Exponent Attacks*) [26].
To generate a key pair, Bob chooses two large (more than 100 bits length) prime numbers p and q, $p \neq q$, and computes

$$n = p * q.$$

n is called the *RSA-modulus*.
Additionally, he chooses a natural number e, the *encryption key*, with

$$1 < e < \varphi(n) = (p-1)(q-1) \qquad \text{with } \gcd(e, \varphi(n)) = 1$$

[3] Clearly it depends on the importance and value of the potential gain.

and computes a natural number d, the *decryption key*, with

$$1 < d < \varphi(n) \qquad \text{and} \qquad d * e \equiv 1 \bmod \varphi(n).$$

In other words

$$d = e^{-1} \bmod ((p-1)(q-1)),$$

which can be computed with the *Extended Euclidean Algorithm*, see [27]. The public key is (n,e) and the private key is d.
To encrypt a message $m \in M$ (M being the message space and $0 \leq m < n$), with a public key, Bob computes

$$c = m^e \bmod n, \qquad \text{with } c \in C \text{ and } C \text{ is the ciphertext space.}$$

If $m \geq n$, m can be split into k subsequent blocks m_i, $i \in 1,\ldots,k$, where each block's length is less than n. Decryption with the private key is computing

$$m' = c^d = (m^e)^d \bmod n.$$

The security of RSA is supposed to be based on the hardness of the factorization problem [19], meaning factoring large numbers. But it is unknown yet if factoring the large modulus n of a key pair is the only way of getting the cleartext message m from the encrypted message c and the public key (n,e), with e being *coprime* to $\varphi(n)$, meaning that $\varphi(n)$ and e have the greatest common divisor 1 [19].
RSA in this form is vulnerable to chosen message attacks. To prevent this, it is advisable to follow the recommendations of the *Public Key Cryptography Standards* (PKCS) devised by the *RSA laboratories* [7]. PKCS#1 describes methods of hardening RSA against the above mentioned attacks sufficiently. RSA is free to world-wide use since the year 2000, as the patent held by the RSA Security Inc. [28] in the United States expired.
Evaluating the costs of RSA, encrypting requires an exponentiation modulo n. Modular exponentiation is performed by a series of modular multiplications. With a smaller exponent e encryption is sped up, but when the exponent is too small, so-called *Low Exponent Attacks* are possible [19]. A common value for e, recommended by X.509 [29], is $(2^{16}+1)$, which is a smart choice because it takes only 17 multiplications to exponentiate. It is possible to choose the same value for e for all key pairs as long as d differs. Decryption is also an exponentiation modulo n, but this time the exponent d has to be about the same size as n. Otherwise, the system becomes insecure. The complexity of usual RSA encryption and decryption implementations is in $O(k^2)$ to $O(k^3)$, with k being key bit length, and key generation is in $O(k^4)$ [7].
Recommended key lengths dependent on the desired usage with respect to the estimated computing capacity of modern computers to break a key pair. Today, key lengths are at minimum 1024 bits and if a key pair should be valid for several years, it is strongly recommended to use a 2048 bits modulus. Again, it depends on the desired usage of the key pair. See Section VI-G for more details. RSA's unproven hardness is its greatest disadvantage. But this can be rebalanced with two arguments: Its widespread use and (because of that) the big interest of cryptanalysts. The more people are interested in the security of RSA (and with it factoring large numbers), the bigger is the

chance that a breakthrough will become public. But being analyzed for over thirty years, the "risk" of breaking RSA seems to be very small. In combination with hash functions (see Section VI-E), known weaknesses like the *existential forgery* and *RSA multiplicativity* become impracticable [19].

2) Rabin: The *Rabin* public key cryptosystem, also named after its inventor Michael O. Rabin, is closely related to the RSA cryptosystem. Its security is also based on the factorization problem, but in contrast to RSA, it is provable equivalent to the factorization problem and therefore considered secure [26]. Encryption with Rabin is a little more efficient than with RSA, while decryption costs are about the same. The disadvantage of Rabin is its vulnerability to *Chosen Ciphertext Attacks*. These are attacks where *Eve* has temporary access to the decryption machine and chooses ciphertexts to get the corresponding messages. With those, *Eve* can compute the private key and totally break Rabin. This is why Rabin has only few significance in practice. In an offline-authentication scheme, such an attack can be accomplished by encrypting a message with the public key and then sending these randomly seeming bytes as challenge to sign. The answer to the challenge *Eve* receives is the decrypted message that can be used to break Rabin. Therefore, it is not reasonable to use Rabin for offline-authentication.

3) ElGamal: ElGamal's security is based on the hardness of the *discrete logarithm problem*. It describes the difficulty of calculating discrete logarithms in a finite field. To create a key pair, a prime p and two random numbers g and x have to be chosen, with g and $x < p$. By calculating

$$y = g^x \bmod p,$$

the public key is (y, g, p) and the private key is x.
Signatures of a message $m \in M$, M is the message space, are made by *Bob* choosing a random number k being *coprime* to $(p-1)$. It is very important that k is random and is never used twice, otherwise *Eve* can recover the private key x [26]. *Bob* computes

$$a = g^k \bmod p$$

and uses the *Extended Euclidean Algorithm* to solve the following equation for b:

$$m = (x * a + k * b) \bmod (p-1).$$

Now, (a, b) is the signature of M; k must be kept secret.
Alice verifies the signature by confirming that

$$y^a * a^b \bmod p \equiv g^m \bmod p.$$

It is necessary for m being a message digest, otherwise messages and appropriate signatures (a, b) can be deduced from another [19]. ElGamal is based on the *Diffie-Hellman key agreement protocol* and the operations to sign and verify are very similar to it [19]. No patents cover its use. ElGamal signing requires computing one modular exponentiation and one *Extended Euclidean*. The calculation of a and y^a are message-independent, they can be computed and stored before the actual verification. Thereby, ElGamal is faster than RSA (with only one exponentiation), but the computations have to be kept secret on the mobile device.

Verification demands three modular exponentiations, two more than RSA. But the verification process can be altered so that computations equivalent to one exponentiation are necessary [19]. Key sizes of ElGamal are about the same compared to RSA for an equal level of security. The ciphertext is double the size of the corresponding message, but this does not matter in our context as long as only message digests and random numbers of relatively short bit lengths are computed. One interesting advantage of ElGamal is the possibility to implement it in any cyclic group other than the prime residue group modulo a prime. If a way was found to calculate discrete logarithms in $(\mathbb{Z}/p\mathbb{Z})^*$, with p prime, then ElGamal could be reimplemented in another cyclic group, where discrete logarithms are still hard to solve [19]. Such a group are elliptic curves over finite fields, discussed in the following Section about an ElGamal derivative, the *Digital Signature Algorithm*.

4) The Digital Signature Algorithm: The so-called *Digital Signature Algorithm* (DSA) is a variant of the ElGamal signature algorithm. It is used in the *Digital Signature Standard* proposed from the United States' *National Institute of Standards and Technology* (NIST) and specified in FIPS 186-2 [30].

To generate a key pair, Bob chooses a prime number q of 160 bits length and a prime p with $2^{511+64t} < p < 2^{512+64t}$ for $t \in \{0,1,\ldots,8\}$. q is to be a divisor of $(p-1)$. Then Bob computes

$$g = h^{(p-1)/q} \bmod p, \text{ with } h \in \{2,3,\ldots,p-2\} \text{ and } h^{(p-1)/q} \bmod p > 1.$$

He chooses a number x with $0 < x < q$ and determines

$$y = g^x \bmod p.$$

The public key is (p,q,g,y) and the private key is x.

To sign a message m, Alice generates a random number $k < q$. Afterwards, she computes the hash $H(m)$ from m with a one-way hash function. The standard specifies use of the *Secure Hash Algorithm* discussed in SectionVI-E. Furthermore, she calculates

$$\begin{aligned} r &= (g^k \bmod p) \bmod q, \\ s &= (k^{-1}(H(m)+xr)) \bmod q. \end{aligned}$$

The signature is the pair (r,s). Bob verifies the signature by computing

$$\begin{aligned} w &= s^{-1} \bmod q, \\ u_1 &= (H(m) * w) \bmod q, \\ u_2 &= rw \bmod q, \\ v &= ((g^{u_1} * y^{u_2}) \bmod p) \bmod q. \end{aligned}$$

The signature is verified by checking that $v = r$, [26].

Because of modular exponentiations of fixed 160 bits length, DSA is faster than ElGamal, which needs modular exponentiations of the length of the module p. Moreover, DSA can be sped up analogue to ElGamal by precomputing the message-independent values r and k^{-1}.

The security of DSA, as an ElGamal variant, is also based on the discrete logarithm problem. One way to attack DSA is the *existential forgery*, which can be prevented by checking

$$1 \leq r \leq q-1 \text{ and } 1 \leq s \leq q-1$$

before verification of a signature. Another way are algorithms alleviating computation of the logarithm problem. The best known algorithms like *Shanks* or *Pohlig-Hellman* [19] still need more than $\sqrt{q}$ steps to solve a logarithm problem of q bits. Because $2^{159} < q < 2^{160}$ in DSA, at least 2^{79} operations are necessary, which can be considered secure [19].
A U.S. patent attributed to David Kravitz, a former NSA employee, covers DSA, but it was made available world-wide royalty-free. Although DSA is efficient, secure and freely available, it is not been integrated in the crypto toolkit, but a variant of it: Elliptic Curve DSA described in the following section, which implements DSA in another cyclic group and offers even more advantages.

5) Elliptic Curve DSA: Cryptographic algorithms basing on the discrete logarithm problem can be improved by implementing them in the cyclic group of elliptic curves over finite fields. By choosing a certain curve, it is possible to apply speed enhancements utilized in $(\mathbb{Z}/p\mathbb{Z})^*$, making signing and verifying faster, but preventing the application of algorithms simplifying the logarithm problem, like *Shanks* or *Pohlig-Hellman* [19].
An advantage of this are smaller key sizes to gain the same security level compared to non-elliptical public key cryptosystems (see Table VIII). The downside are more complex mathematical computations for generating signatures, so that implementations of elliptic curve cryptosystems in spite of smaller keys are not necessarily faster.
Elliptic Curve DSA (ECDSA) is a variant of DSA based on the *Elliptic Curve Discrete Logarithm Problem* (ECDLP). Because elliptic curves were of interest in mathematics a long time before the application to cryptography was considered, the risk of developing new algorithms breaking the ECDLP is reduced.
Elliptic curves do not represent curves or even ellipses in the common sense, but are points solving the equation

$$y^2 = x^3 + ax + b.$$

Selecting an appropriate curve requires a good understanding of the mathematics of elliptic curves and is troublesome to implement. Therefore, the NIST published recommended domain parameters of elliptic curves to use [31]. The choice of a certain field is significant for overall performance, so for implementation, finite fields of odd characteristic ($\mathbb{F}_p$, where $p > 3$ is a large prime number) and fields of characteristic two, $\mathbb{F}_{2^m}$, are considered best [32].

To create a key pair, the domain parameters (q, FR, a, b, G, n, h) of the underlying curve E have to be chosen:

- p specifies a prime power,
- FR describes the method of representing field elements $\in \mathbb{F}_q$, with $q = p$ or $q = 2^m$,
- a, b are two field elements $\in \mathbb{F}_q$ specifying the equation of the curve,
- G is the base point $G = (x_G, y_G)$ on $E(\mathbb{F}_q)$,
- n is a prime of the order of G,
- h is an integer, which is the cofactor $h = \#E(\mathbb{F}_q)/n$.

The private key is a random integer $d \in [1, n-1]$ and the public key is $Q = dG$.
Alice signs a message m by selecting a random integer $k \in [1, n-1]$ and calculating

$$r = x_1 \bmod n, \text{ where } (x_1, y_1) = kG,$$
$$s = k^{-1}(H(m) + dr) \bmod n,$$

where $H(m)$ is the hash value of m. If $r = 0$ or $s = 0$, she has to select another random number and compute the signature again, otherwise, the signature is the pair (r, s).
For verification, *Bob* checks that r and s are integers in $[1, n-1]$. Then he hashes the message m to obtain $H(m)$ and calculates

$$\begin{aligned} w &= s^{-1} \bmod n, \\ u_1 &= H(m)w \bmod n, \\ u_2 &= rw \bmod n, \\ (x_1, y_1) &= u_1 G + u_2 QA. \end{aligned}$$

The signature is valid if $x_1 = r \bmod n$.
ECDSA over $\mathbb{F}_p$ is considered secure with key lengths of 192 bits, in contrast to DSA, which needs key lengths up to 1024 bits or RSA with more than 1024 bits.

G. Summary

The focus of these examinations is to provide an overview and a brief background as well as evaluation of digital signature algorithms to be used for *incremental contract signing*. If possible, encryption functionality by the use of these algorithms should also be available. It became clear that only two algorithms, RSA and ECDSA, fulfil the prerequisites and can be used. Both are also recommended by the German government organization BSI[4] in their annual recommendation on qualified electronic signatures [33]. Although ECDSA has been designed as a digital signature algorithm, its use for encryption has already been proposed. But, no detailed information on an successful employment are available. Both, RSA and ECDSA can be adapted within a certain range by adjusting the key lengths to the needs imposed by the setting, e.g. the desired cryptographic strength. While the key length of RSA can be changed easily, the adaption of the ECDSA key length needs more profound knowledge on elliptic curve groups. Although recommendations which parameters to use exist, the choice is limited. Especially the lower end of cryptographic strength, which can be addressed by the use of RSA in an easy manner, is not covered.

The website keylength.com [34] is a project by BlueKrypt [35] providing recommendations on cryptographic key lengths based on different sources – most of them are government agencies and research groups. An easy to use comparison of the slightly varying recommendations and emphasis is provided on their website together with links to surveys addressing laws and regulations on cryptology. Other publications such as the *NIST special publication SP800-57 part 1* addressing computer security [36] and the *NSA Suite B Cryptography* [37] have a much broader scope, but give similar recommendations on cryptographic algorithms and key lengths to be used for different scenarios. The European IST research project ECRYPT releases their "Yearly Reports on Algorithms and Keysizes" [38] covering a broad range of cryptographic algorithms and related topics. Table VIII contains an excerpt of the relevant information, such as key length of comparable strength and recommended usage scenario. Key lengths of symmetric cryptography algorithms of equivalent key length are provided for comparison.

TABLE VIII
KEY LENGTHS [BIT] OF EQUIVALENT SECURITY

ALGORITHM	RSA	ECDSA	Symmetric
STRENGTH	1024	160 - 223	80
	2048	224 - 255	112
	3072	256 - 383	128
	7680	384 - 511	192
	15360	512+	256

A rough classification is given in [38] stating that RSA1024, which will later be recommended as least demoninator, will provide "Very short-term protection against agencies, long term protection against small organizations", RSA768 in contrast only "Very short-term protection against small

[4]BSI: Bundesamt für Sicherheit in der Informationstechnik

organisation" which we also consider secure enough for trading assets of small value by the use of *incremental contract signing*. By the way, RSA15424(!) should be a good choice for the "foreseeable future", which means good protection against quantum computer. Due to the recent involvement of massive parallel computations running on graphic cards, normally used for complex scene calculations, further estimations on the time needed to break a key and its security are hard to provide. The beginning of this development can be found in Table IX which has been derived from RFC 4359 [39] during its specification process [40]. The RFC itself declares support for 1024 bit key length as mandatory, neither listing key lengths below 768 bits nor above 1024 bits. This trend is even accelerating due to the availability of faster hardware and more sophisticated usage. These information are also used by [38], but stating that they would already contain a 100-fold security margin.

TABLE IX
RSA KEY LIFETIME RECOMMENDATIONS.

STRENGTH	MAXIMUM LIFETIME RECOMMENDED	
Key length [bit]	in 11/2004	in 01/2006
496-512	1 hour	-
576	10 hours	-
640	4 days	-
704	30 days	-
768	-	1 week
796	8 months	-
1024	1 year	1 year

Therefore it is reasonable to use key lengths of at least 768 bit. We recommend and will use throughout our research key length of 1024 bit. Of course, it depends on the setting, such as the time span and the number of times the key pairs should be used. If a usage of multiple years is desired, much longer keys have to be used as for key pairs suitable for one-time usage. Another factor is the expiration date of the contract, i.e. the time is span it can be submitted for processing. We assume a maximal contract life-time, which is the time from signing the contract until its submission, of at most a couple of weeks. Last but not least, the value of the traded assets are also of importance, because if the contract addresses high priced assets, it might be worth an huge effort. Although such adjustments are possible we will stick to RSA1024, as it turned out, nowadays devices are capable of real-time calculations and key pair generation up to 1024 without any restrictions. But more important it fits our proposed usage with respect to the application and assets to be traded.

VII. Real World

In this section we will first provide an overview of the mobile phones, platforms and libraries that were employed. Then, the applications and scripts used to derive as well as to process the measurements are introduced. Measurements, criteria for plausibility checks as well as observations and finally an evaluation will be provided.

A. Devices

We were able to get hands on devices from several vendors equipped with different processors and operating systems, but all of them are capable of our prerequisites: the CLDC 1.1, the MIDP 2.0 and the JSR-82. A short overview and characterization is given in Table X. The cell phones are dedicated to different segments of the market *a)* "usual" just-a-phone, *b)* phones with enhanced multi-media and photo functionality and *c)* finally so called "smart-phones" with focus on office-functionality.

TABLE X
Considered Mobile Devices.

Vendor and Type	Classification at time of release	OS	CPU	RSA	ECDSA
Sony Ericsson K750i	Photo Q2/2005	proprietary	?	×	?
Siemens S65	Business Q3/2004	proprietary	?	×	∅
Sony Ericsson M600i	Smartphone Q2/2006	Symbian 9.1 UIQ 3	ARM9 based, 208 MHz	×	×
Nokia E50	Smartphone Q3/2006	Symbian 9.1 Series 60 3rd	ARM-926, 235 MHz	×	×
Nokia 5800XM	Multimedia Q4/2008	Symbian OS Series S60 5th	ARM11, 369 MHz	×	×
Nokia 6300	Usual Phone Q1/2007	Symbian OS Series S40 3rd	ARM9, 238MHz	×	∅

The two older ones, K750i and S65, with their small displays and little multimedia functionality, are already obsolete. Although RSA is possible on the K750i, the time needed is not acceptable. While the K750i is not able to do ECDSA processing or does need too long without responding; the S65, which was targeted at the business sector at that time performed RSA and ECDSA processing despite its age quite well. Both devices are not included in the test series as they are not widespread used any more and do not represent current classes of devices. The M600i and the E50 are two smartphones with office as well as multimedia functionality released in 2006. Still, both are quite performant. The 6300 is a consumer cell phone released in early 2007. Processing ECDSA fails with an exception, but due to mandatory obfuscation of the code debugging has not taken place so far. Its RSA performance is quite disappointing. Although the recently released 5800xm offers all kinds of multimedia, GPS and communication ability, its performance is similar to the smartphones that have been released about 30 months ago. This set of devices covers a large range of similar devices already in use and provide a good estimation of the available potential waiting to be used.

B. Libraries and Software

This report focuses on the mobile side of the digital ownership framework, as computational power and libraries providing access to cryptography are available at the PC. The aim is to enable mobile usage of strong cryptography on widespread used and inexpensive devices. We have chosen the freely available *openssl* library together with its ruby-binding *libopenssl-ruby* for the implementation of the central server. The web front-end and API itself is realized by the use of Ruby on Rails. All necessary software is available for Linux – we use kubuntu, debian and Mac OS X for our development. Only cell phones that provide the necessary prerequisites without enabling cryptography itself have been chosen. The entire support for cryptography and other convenient functionality such as easy usage of persistent storage and communication had to be realized by own libraries.

A library called *CryptoToolkit* has been implemented, making use of the recent version of the Bouncy Castle libraries [8]. The entire cryptographic functionality, needed by the MIDlet employed for these tests and our other implementations within the digital ownership research, have made accessible by its APIs. Such high-level functionality includes key pair generation, signature creation and verification as well as hashing. Besides this library, *MEmFiS* (short for *M*obile *Em*ulated *F*ile *S*ystem) has been used to store any kind of data, especially measurements, persistently. Its add-on *MEmFTP* enables the access to the data files stored within MEmFiS from the PC via Bluetooth in an FTP-like manner. The ability to access the "normal" file system inside the phone and memory cards is subject of ongoing work. Convenient discovery and communication functionality between multiple mobile devices and personal computer has been implemented in the *BtCom* library.

The applications for the performance measurements of cryptographic operations and the communication using Bluetooth have been designed in a straight forward manner by the use of JavaME's high-level API. The different algorithms and key lengths, e.g. RSA512 with SHA-1, to be tested can be selected and the number of iterations as well as a description can be set. A test is a combination of a hash-function and a digital signature algorithm or a hash-function with a certain payload size. Then, the tests run interleaved, i.e. each test once per iteration to even out short, timely limited disturbances. Additional tests of the communication between two devices have been implemented in a separate application. Both applications store the test results in a dedicated directory with a timestamp as name inside the persistant memory.

C. Test-Setting

All devices, except the two obsolete and not further considered ones, were equipped with a valid SIM card of the same GSM-network, the one of the German provider ePlus. During the test all devices were located side by side within an area of about 0.25sqm, i.e. the network coverage and the climatic conditions should have been the same. First of all we were curious as devices of different generations and market segments were involved. During our early tests the following two important factors have been identified: *1)* the current computational load and *2)* the availability of an external power supply. Besides this, the influence of the communication modules (such as GSM, Bluetooth and wireless LAN) should also be evaluated. To avoid any further disturbance the screen saver has been disabled, but reduction of the backlight's brightness remained enabled to save battery power. Unfortunately, no information about the precision of the built-in clocks are available and no correction will take place. Normally, temperature has an impact on two important sources of disturbance, the quartz driven clock and the accumulator. As precaution, the mobile phones where kept at approximately the same temperature avoiding exposure to cold air and direct sunshine. Also, no information on the power saving strategies, i.e. if and how does the device react if running on battery and what happens if the remaining battery capacity decreases, are available.

As a consequence of these considerations and the first test runs, the following factors have been identified:

1) the availability of an external power supply,
2) the status of the communication modules (turned off or being idle) and
3) the current computational load imposed on the device.

Their potential influence will be examined and rated later on. While the first two ones are clearly defined for each mobile, the third one, the computational load, is quite device specific: from idle except the speed-test MIDlet running to the device specific full-load. The latter one ranges from the load produced by simple playback of audio files in the background (lack of real multitasking functionality) to simultaneous WLAN access, audio playback, and finally GPS running in the foreground. It turned out, that no generic multitasking functionality is available on all devices, e.g. the music player is the only application on the 6300 able to run in the background.

The following scenarios have been examined with different intensity:

TABLE XI
FIVE SCENARIOS TO ADDRESS DIFFERENT FACTORS OF INFLUENCE.

No.	power supply	load status except measurement	communication modules	primary issue of this scenario
1	external	idle	online	basic setting
2	external	idle	offline	influence of (idle) communication modules
3	battery	idle	online	scenario 1 w/o power supply
4	external	full-load	online	influence of massive computational load
5	battery	full-load	online	scenario 4 w/o power supply

The *powered idle online [1]* setting has been chosen as reference. It allows precise measurements of the device's performance when processing the cryptographic operations within a realistic scenario, but with as little disturbance as possible. No user triggered communication takes place. Then the tests were run in the offline mode to estimate the influence of the communication modules *[2]* and unpowered to figure out if there are any power-supply *[3]* related changes. In case of out-of-time operations which can happen during nighttime – such as key generation or certificate creation whose results can be held on stack – the powered online idle scenario *[1]* is not unlikely. So far, the device had been dedicated to our measurements, which means “doing nothing else”. Two more scenarios are examined where a device specific full-load is imposed – online with and without external power supply. If possible, our calculations were not the focused application in the full-load scenarios.

A single test consists of a combination of a hash function (SHA-1, SHA-256, SHA-384 or SHA-512) in combination with a digital signature algorithm, a choice of RSA (with key lengths of 512, 768, 1024, or 2048 bits) or ECDSA (with key lengths of 192, 239 or 256 bits). During each test, *1)* one key pair and *2)* one hash sum to be signed have to be created. Then, *3)* the signature is calculated for the hash sum by the use of the just generated keypair and *4)* thereafter verified.

The runtime for each of the above mentioned operations are measured in milliseconds and stored. Due to very short execution times of hashing and verification, multiple repetitions of each operation are carried out inside the chronometry to achieve more reliable measurements. During an iteration each of the above mentioned algorithm/keylength combination can be run zero or one time. A test series itself consists of a number of iterations, which has to be chosen according to the scenario, especially if running on battery. While powered runs are not restricted to a certain number of iterations, the maximum number of iterations in the unpowered test series depends on the imposed load and the capacity of the battery which lasts between 7 and 15 iterations. Figure 13 shows a sketch of these dependencies.

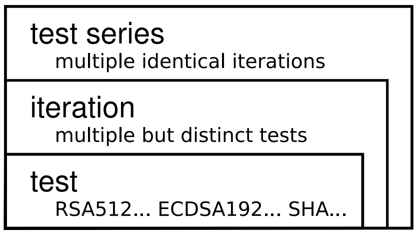

test series
multiple identical iterations
iteration
multiple but distinct tests
test
RSA512... ECDSA192... SHA...

Fig. 13. Cryptographic Measurements: Test, Iterations and Series

Test results are kept in the RAM until the end of the test series and stored as comma separated values. This leads to an increasing memory consumption with each test. The measurements are stored inside the record store after each iteration in case of a device failure – most likely an empty battery or on the M600i a self-triggered reboot due to the video player running for a while. The transfer to the PC takes place by the use of Bluetooth. The device has to be restarted before each test series.

D. Processing and Quality control

A number of scripts have been written to allow automatic processing of the measurements and the derivation of the most important statistics once the test results have been stored on the PC. Below the project directory, a subdirectory for each device is mandatory as some device specific configurations have to take place on the one hand and on the other hand to clearly separate the findings. The following steps have to take place:

1) The results of each test series are stored in a separate file whose name indicates its time of execution, the device, the number of iterations, the algorithms and most important the scenario it took place.
   E.g. `1239056923655_5800xm_025ers_online_powered.csv`.
2) Multiple test series are merged to a device specific scenario file, e.g. *online_powered*, *fullload_unpowered*, etc..
3) Scenario files are analyzed and relevant information are extracted and stored for further processing, i.e. statistics and histogram data are generated and plotted.
4) Temporary data files are merged to allow for inner and outer ratio estimation.
5) Based on these information further manual analysis, such as the examination of the timely development of single series or even algorithm and key length combinations can take place.

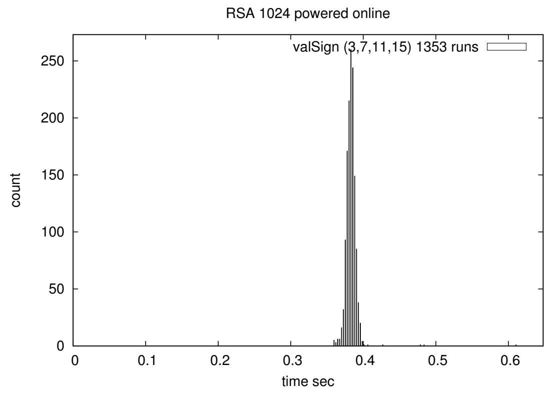
Steps 1-4) have been automated to gather the measurements to derive the ratios and statistical information. Histograms and cumulative distribution function have been generated to rate and to compare the single scenarios with others. Further statistics such as error bars are also processed. Plotting individual runs is possible by the manual usage of the same command-line scripts. Besides this, the measurements of a single test series can be plotted in a chronological order.

As the cell phones are more or less black boxes, the quality of the measurements can only be guaranteed by thoroughly observations and precise definition of the setting. First of all, our measurements were derived from restarted devices to get rid of reduced random access memory due to memory leaks and further influence that might arise from long uptime and applications that have run before. Then, only a defined set of applications runs under the same conditions.

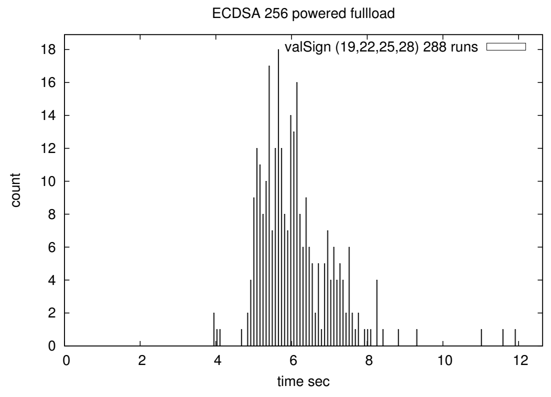
Quality control refers on the one hand to detecting problems with the setup itself or during interaction with the devices, e.g. power supply is still attached or a media player has been paused due to task switches, and on the other hand to figure out if there are any obvious disturbances resulting from running the measurements. This could be odd behaviors such as increasing execution times during test series, completely different results of "similar" set up test series, etc.. No further countermeasures can be taken, with the limited insight in the device internals, besides: restart, thoroughly configuration and providing similar conditions for each test run. After the transfer of the test results, automatic processing will take place. Besides this, an individual examination of each test series is necessary to understand plots such as shown in Figure 14(d). Otherwise it can not be distinguished between two peaks arising during a single test series or two independent peaks, each one from a different test series.

In Section III-E the runtimes of the shortest key for each algorithm, RSA512 and ECDSA192 were provided. These information have been derived from our set of measurements. The statistical information are available for all devices and scenarios as error bars providing minimal, maximal, average, median as well as the standard deviation. Due to the vast amount of plots, only the two extreme scenarios discussed before are contained in Appendix A. The histograms of the execution times for every operation, device, algorithm and key length combination can be found in Appendix B.1 and B.2. They are grouped by device, scenario and algorithm. Based on these histograms the distribution of the runtimes can be estimated to allow prediction of the runtimes for specific settings. A third type of measurement presentation are the cumulative distribution function provided in this section for two exemplary combinations of algorithms, but all devices. They allow a much better insight into the distribution and enables an easier comparison of the five scenarios.

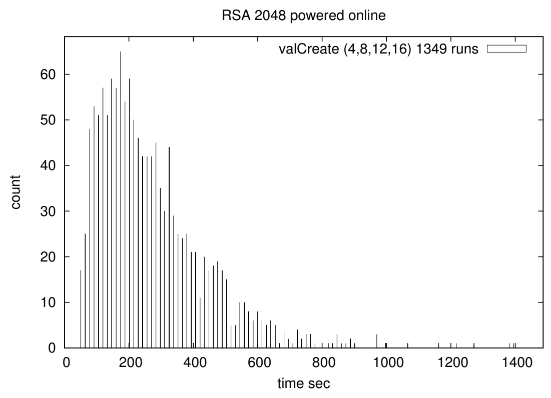
Our expectations were that a single, precise and symmetric peak in the histogram, such as shown in Figure 14(a), would exist for the deterministic operations (all except RSA key generation). In case of full-load scenarios blurry, but almost symmetric "peaks" seemed to be reasonable. This is an indication for a higher standard deviation due to multiple tasks handled in parallel. An example for this effect can be be found in Figure 14(b). In offline scenarios, where less load has to be handled, and in the online unpowered, where processor speed might be reduced, the expected peaks could have an offset to either side, due to shorter resp. longer execution times. Due to the indeterministic nature of prime number search, RSA key generation does not result in a symmetric distribution, visualized as a peak, but in a maximum with a descending slope to its right – as shown in Figure 14(c). Our expections became true and we derived these patterns in various settings and for different devices.



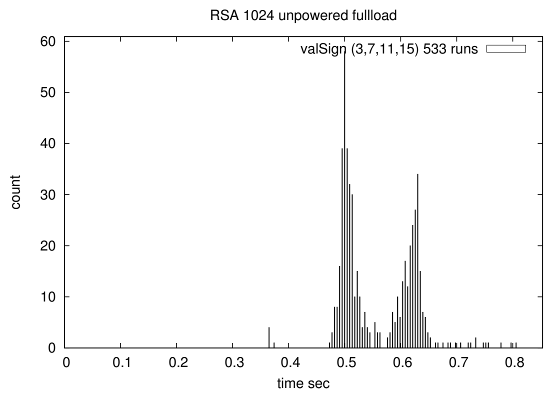
(a) precise, single peak.



(b) blurred, single hill.



(c) slopy hill.



(d) multiple peaks.

Fig. 14. Examples for Histograms.

Histograms are good to get a first idea of the situation on the one hand and on the other hand they allow for the comparison of single test series to learn about (self) similarities. This has been used to analyze the situation shown in Figure 14(d) as these two peaks can *a)* exist in any test series or *b)* it is composed of two single peaks appearing in different test series, but with an offset. It turned out that the vast majority of our test series contained two peaks, a few had only a single peak. It was not possible to determine the reason. The self similarity has also to be checked, i.e. if a blurred "peak" in a scenario files emerged from multiple test series results from multiple narrow but slightly shifted peaks or from several identical blurred peaks. The aim was to get an impression of the devices behavior and of the test series to rate the reliability of the findings and recommendations.

E. Observations and Findings

There are several approaches to compare entire scenarios as well as single test series and the timely development of a certain test during an entire series. As mentioned before, histograms have been generated to visually analyze the distribution of the runtimes and compare different test series of the same scenario. Four exemplary histograms are shown in Figure 14.

Now, we will describe the observations we have made. Later on, they will be rated according to their severity. Three basic plot types have been used and examined to figure out any oddities: *1)* histograms of the RTTs have been generated for each test series and their aggregate according to device, algorithm and key length, *2)* certain series have been plotted according to their timeline and finally *3)* the cumulative distribution function to allow for better comparability between single scenarios. While the first two ones are primarily used to assure the quality of the measurements and to learn about potential pitfalls, the last one, the cumulative function, can be used for direct comparison of the five scenarios according to the three identified factors, power supply, communication modules and the load imposed on the devices. We will refer to the cumulative functions of RSA1024, representative for RSA, shown Figure 15 and ECDSA192 in case of ECDSA as presented in Figure 16. Each row of plots is dedicated to a device, the columns represent the three cryptographic operations: key pair creation, signature creation and its verification. Key pair generation is measured in seconds, signature creation and verification in milliseconds.

Now, it is possible to directly compare the five different settings by the use of the provided cumulative curves for RSA and ECDSA. Although our main focus is clearly on RSA the findings in this section will also consider ECDSA. The three main factors that influenced the scenario creation are discussed in the following.

power supply. Obviously, cell phones are designed for mobile usage without being connected to a(n external) power supply. Because of this, the main functionality – telephony – should not be affected if running on batteries. In contrast, notebooks suffer the problem, that as soon as they run on batteries, either the available computational power or the lifetime of the battery decreases significantly. Modern mobile devices are trying to combine the functionality of both sides. Fortunately, the observed influences are less dramatic – but also surprising.

The M600i is the only device where a significant effect can be observed. It is faster in every unpowered scenario regardless of processing RSA or ECDSA. The E50 also behaves quite strange as only RSA key pair generation is speed up by running on batteries, no difference can be detected for the other RSA operations. In case of ECDSA it is not that explicit, as there is an insignificant shift. It seems that the unpowered devices have a smaller variance of the runtimes. The behavior of the 5800xm is consistent across both algorithms, because the device is faster in the unpowered full-load than in the powered scenarios. Due to the inability to do ECDSA, only RSA measurements can be evaluated for the 6300. It turned out that the variance is small compared to the other devices for every scenario. Furthermore, for both extremes of computational load, idle and full-load, the curves representing scenarios with and without external power supply can not be distinguished.

Unfortunately, we do not have an explanation for all these different behaviors. Our expectation was that running on batteries would result in a worse performance, but in some cases the opposite turned out to be true.

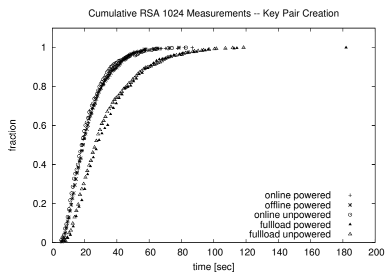
$\Rightarrow$ Not being connected to a power supply cannot be considered to cause any performance reductions. Instead a slight speed-up has been be detected for some devices.

computational load. The device specific full-load is of interest as it is most likely that the device will not be entirely dedicated to our calculations during interaction with others. Showing off the belongings, play back of content, as well as communication will take place. The impact of additional computational load is clearly visualized in Figure 15. The respective scenarios are clustered. Table III provides some numbers for the two most extreme settings representing these clusters and the algorithm and keylengths combination that have been chosen as unit. Two completely different curves appeared in the RSA full-load scenarios: *1)* in case of the E50 and the 6300 an almost vertical and point-symmetric one with no difference between powered and unpowered and *2)* in case of the M600i and the 5800xm two convoluted curves, partially with steps and obvious difference between unpowered and powered. The individual full-load was for the E50 and the 6300 playing back music, while the M600i played back a video in the foreground and the 5800xm was mainly busy with GPS and playback of music. Things are quite different if ECDSA is used as only the idle scenarios of the 5800xm provide almost point symmetric curves, i.e. little variance. Again, the scenarios are grouped by the computational load and except the just mentioned exception almost identical for each of the three operations. Unfortunately we can not provide an explanation for the higher variance even in idle settings.

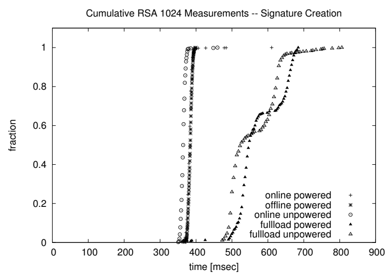
$\Rightarrow$ Additional computational load has a strong impact on the performance of our application, i.e. the runtimes for the cryptographic operations, ranging from additional 30% to about 80%.

communication modules. There was only one out of five scenarios where the communication modules were turned off. Based on the measurements we could not find a significant difference between idle and turned off communication modules in our measurements. Unfortunately, we were not able to carry out tests with data massive data transfer imposed on the communications modules.

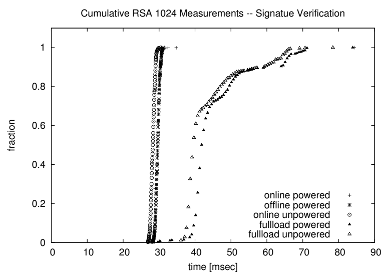
$\Rightarrow$ No influence of (idle) communication module has been detected.



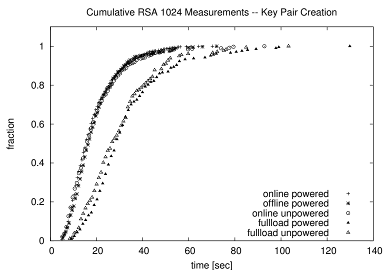
(a) M600i.



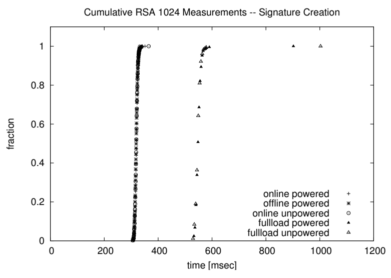
(b) M600i.



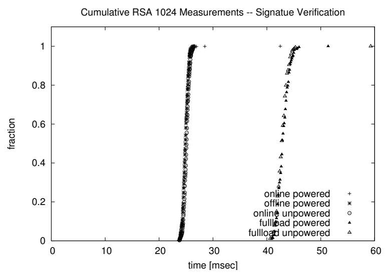
(c) M600i.



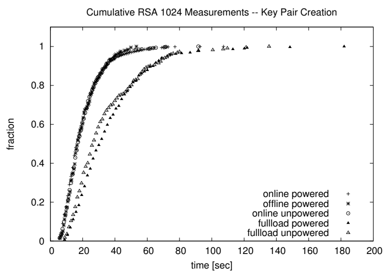
(d) E50.



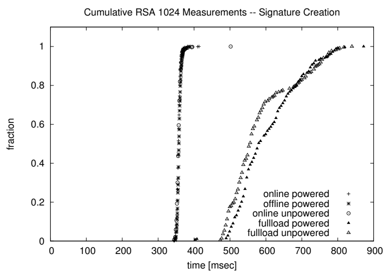
(e) E50.



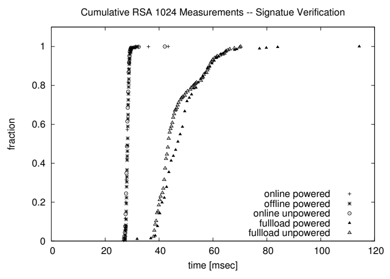
(f) E50.



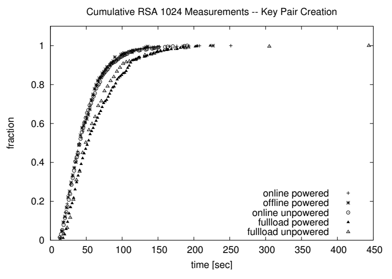
(g) 5800xm.



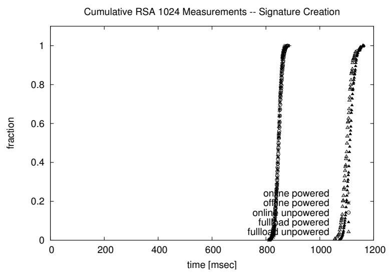
(h) 5800xm.



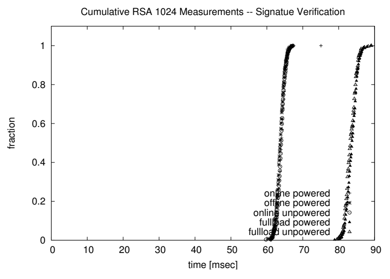
(i) 5800xm.



(j) 6300.

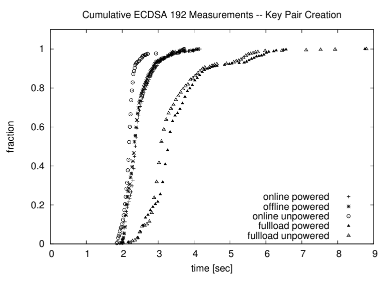


(k) 6300.

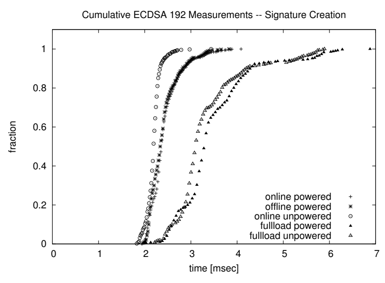


(l) 6300.

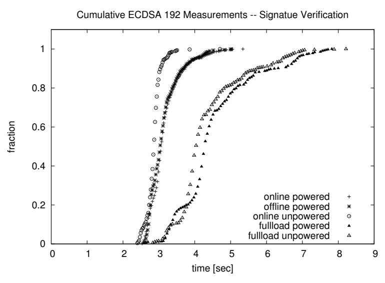
Fig. 15. Cumulative, RSA1024, Key Pair Creation, Signature Creation and Verification.



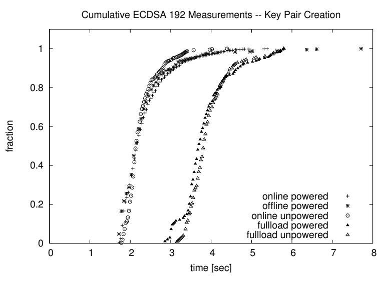
(a) M600i.



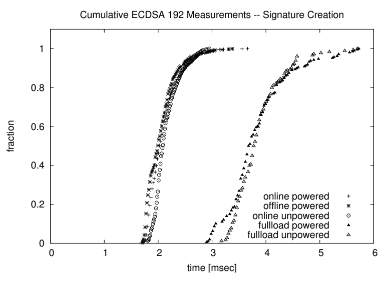
(b) M600i.



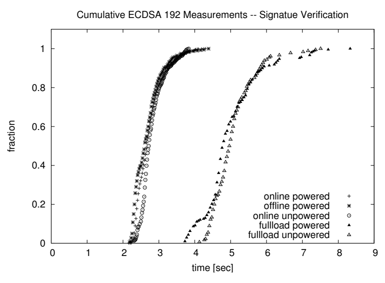
(c) M600i.



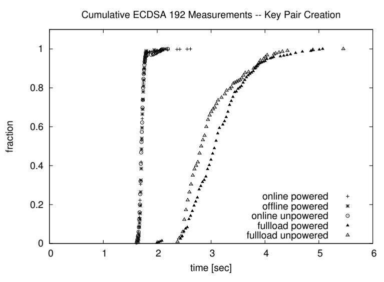
(d) E50.



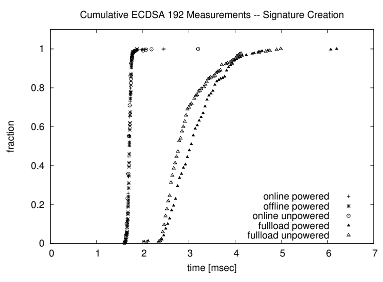
(e) E50.



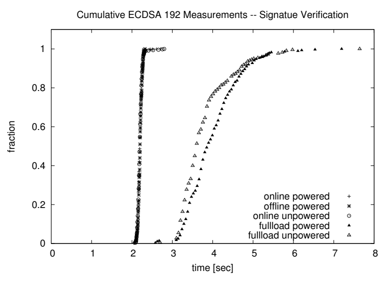
(f) E50.



(g) 5800xm.



(h) 5800xm.



(i) 5800xm.

Fig. 16. Cumulative, ECDSA192, Key Pair Creation, Signature Creation and Verification.

The severity rating of the three examined factors is straight forward. Obviously, the computational load has the biggest impact on the performance of every user installed application. Telephony, other services and media playback seem not to be affected by the high computational load. It is most likely that the build-in task scheduler is employed with a priority ranking of the applications. Our measurements revealed that the performance loss for our third party application will be at least 30%, depending on the device. In the worst case a runtime increase of 80% has been experienced. Therefore, it is strongly recommended to integrate performance measurements into every application to learn about this very device to be able to adjust the own settings and expectations.

The remaining two factors have a slight positive or no detectable impact on the performace. Surprisingly, the absence of an external power supply caused a slight performace gain for some scenarios and devices. Indeed, this is quite odd and we do not have an explanation, but there is no other solution than to charge the own device from time to time. Therefore it is not regarded as a problem.

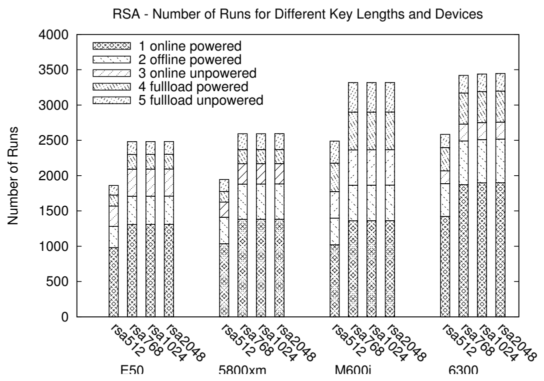
In case of the communication modules being idle or turned off, no difference has been detected. Massive data transfer might have an impact as the transfered data has to be processed by some application. Then, the performance reduction will at least result from the application, which is comparable to the full-load scenarios. Therefore, only the computational load imposed on the device can be named as a performance reducer.

F. Quantity of Measurements

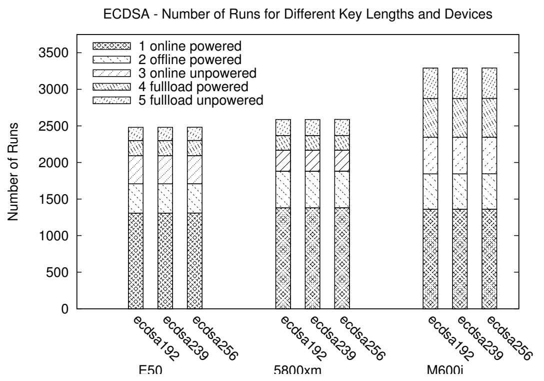
This section will provide information about the quantity of tests that have been accomplished. Due to the different number of devices capable of the two different algorithms and different number of keylength to chose from, about 44.000 RSA and 27.000 ECDSA runs took place. The distribution across devices is depicted in Figure 17(a) and 17(b). The main focus and was on idle online scenario to get a solid set of reference data. Because of this, the majority of the test runs has been dedicated to this scenario. By comparing the merged scenarios with the single test series it became clear that, due to their similarity, a sufficient number of test series have taken place. Besides the number of runs used to derive our findings, the number of failures are also of interest. During the test runs, only the final result of the signature verification, i.e. success or failure, is also recorded. As no intermediate results such as key pair, hash sum and signature are stored, none of the single operations can be identified as the source of the failure. The following causes of failures can be named: hardware fault, operating system or transmission failure, malicious implementation of the application and in case of RSA key pair creation.

Although ECDSA key generation should never fail, we experienced two runs (on different devices) where signature verification was not successful. If this is set in relation to the overall number of test, our implementation achieved a probability of 1:13500 for a failure. Unfortunately, the point of failure can not be identified as no intermediate results between each operation are stored. Because the source code of the Bouncy Castle libraries are freely available and the project itself is widespread in use, an erroneous implementation should be considered very very unlikely.

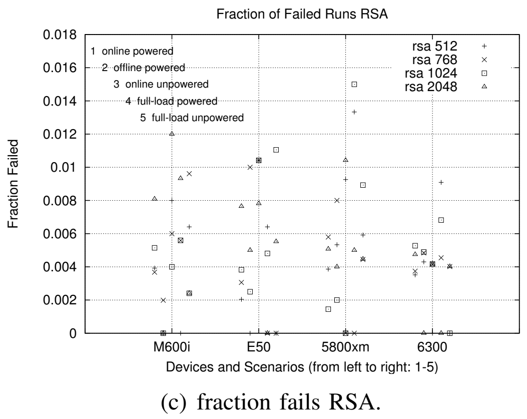
In contrast, RSA key pair generation relies on finding of a real prime number, i.e. failures have to be considered. Prime numbers are created using the Bouncy Castle library with a prime number certainty of 25. This certainty means that there is a probability of $(1-(\frac{1}{2})^{25})$ or ≈ 0.9999999702 that the number selected is truly prime. The propability of a failure is according to the parameterization about $1 : 3 \cdot 10^{-}8$. It turned out that this probability has never been met for any keylength on any device. Figure 17(c), which is quite cumbersome to read, shows the probabilities. The point type indicates the key length; the scenarios (1-5, see Table XI) are plotted within a device from left to right. In some settings up to 1.5% of the runs at least one of the following operations failed somehow: creation of a keypair, signature creation and verification as well as persistent storage and transfer. The failures occurred among several test series. Due to the limited number of tests, it is hard tell what went wrong. As all devices are affected in a similar manner, except the 6300 where the failure rates of the different combinations are closer together, it seems to be not a device specific issue. If the application and the device is considered as correctly functional, the desired propability that the chosen prime number is definitely prime does not hold. A thoroughly examination of this aspect was done in [4]. Due to this much higher rate of invalid key pairs, our library checks each key pair by the use of a similar process as used for our measurements before it is released to the public. This is, according to additional costs for signature creation and verification, absolutely feasible. As key pair generation is not likely to happen just-in-time, this high fraction of broken key pairs can be accepted.



(a) Number of RSA Test Runs.



(b) Number of ECDSA Test Runs.

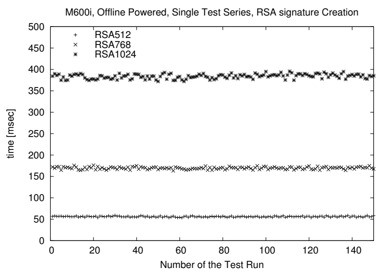


(c) fraction fails RSA.

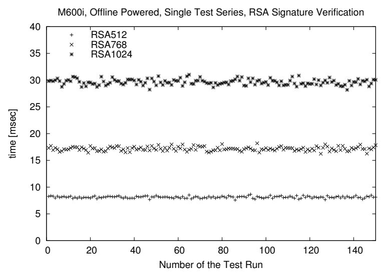
Fig. 17. Number of Test Runs and Fraction of Failures. RSA and ECDSA

G. Odd Observations

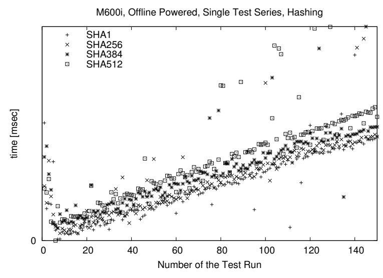
As mentioned before, we examined several test series with respect to the timely development of the measurements, i.e. we want to detect if former test runs influence the later ones. Fortunately we did not detect any indicators for self-influence of the test series in case of the cryptographic operations. But we noticed an increasing runtime of the hash functions within a single test run, which can be seen in Figure 18. For two series are shown runtimes for RSA signature creation, its verification and hashing is shown: The M600i offline powered scenario plots in Figure 18(a), 18(b) and 18(c), the 6300i offline powered signature plots in Figure 18(d), 18(e) and 18(f).



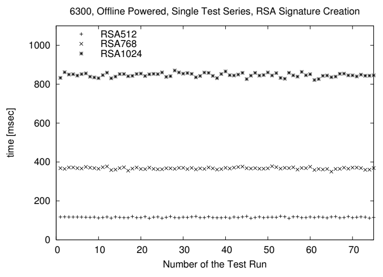
(a) M600i, Signature Creation.



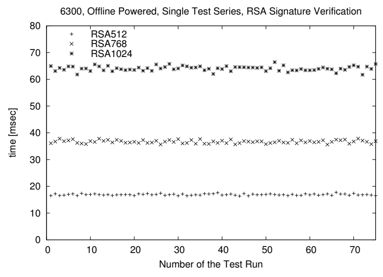
(b) M600i, Signature Verification.



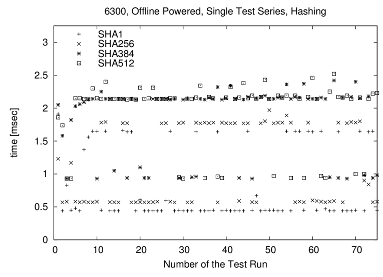
(c) M600i, Hashing.



(d) 6300, Signature Creation.



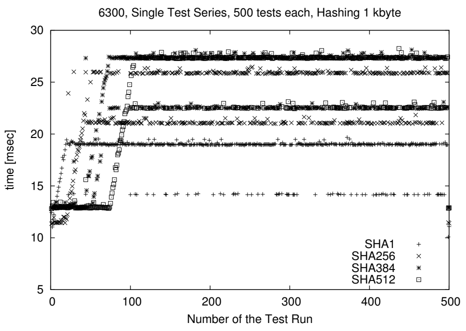
(e) 6300, Signature Verification.



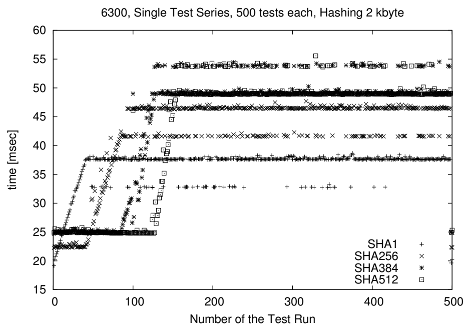
(f) 6300, Hashing.

Fig. 18. Execution Times of a Single Test Series, RSA, Offline Powered.

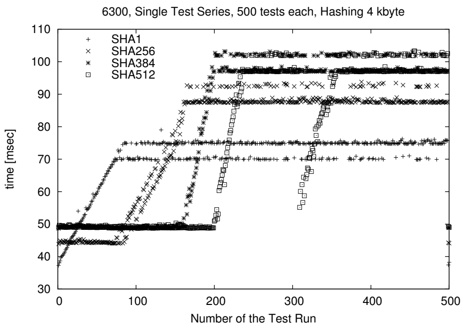
In contrast to the runtimes of signature creation in both settings, the runtimes measured for hashing result in unexpected plots. In case of the M600i the runtimes increase linearly after a few higher values in the beginning. Both facts are surprising, first of all the obviously increasing runtime and the first few higher values for each algorithm in the beginning. We don't have an explanation for either of the phenomenons. The latter one might be due to initial class loading. Hashing in the 6300 scenario results in a quite different plot. Instead of linear increments each hash algorithm has two "preferred" runtimes. After these effects had been discovered a thoroughly examination of the measurements took place. Such effects where only found for the hash functions, which might be due to the short runtimes compared to the cryptographic operations. Because of the small amount of data to be hashed, dedicated test series for hashing bigger chunks of data had been carried out. While the 5800XM did not show any oddities, the 6300 produced strange and hard to interpret measurements, which can be found in Figure 19. Each of the six subfigures 19(a) to 19(f) is dedicated to a certain payload, but each of the four hash algorithms has been carried out for every payload size. During each iteration, each hash function was run for every payload, i.e. both hash functions and different payload sizes were interleaved. Because of this, longer lasting or severe changes should appear in all six plots during the same iteration – which was not the case. The plots turned out to be quite surprising. We identified constant periods, slow as well as fast linear increases as well as fall-backs – even within the same test. Clearly, the runtime of the hash function depends linearly on the amount of data to be hashed, which can be verified by comparing the range of the subfigures y-axis. The runtimes vary by a factor of 6 for the majority of the tests. This can become an issue if bigger amounts of data have to be hashed. Unfortunately we don't have an idea what caused this strange behavior. But, as this has only a minor impact on the feasibility of *incremental contract signing* we postpone further examinations.



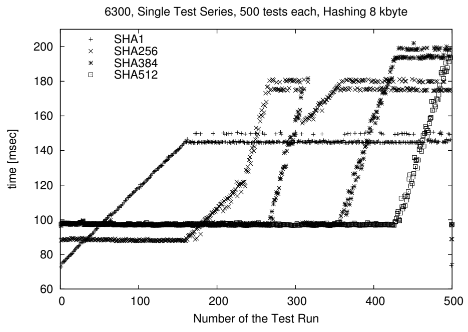
(a) 6300, Hashing 1 kilobyte.



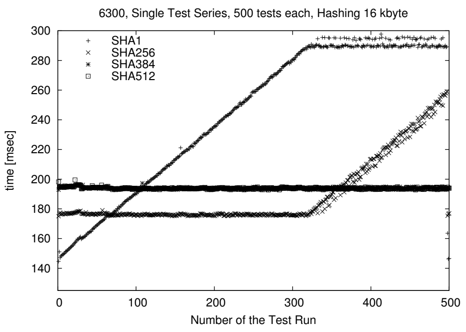
(b) 6300, Hashing 2 kilobyte.



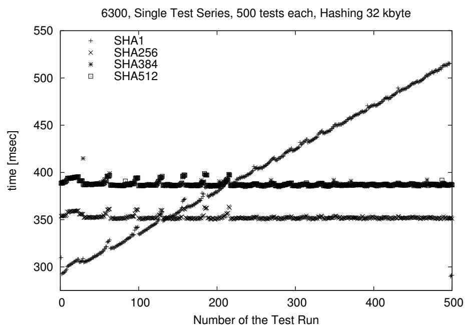
(c) 6300, Hashing 4 kilobyte.



(d) 6300, Hashing 8 kilobyte.



(e) 6300, Hashing 16 kilobyte.



(f) 6300, Hashing 32 kilobyte.

Fig. 19. Executions Times of Hashing Different Payload Sizes.

VIII. Conclusion

This technical report provided the evidence that *incremental contract signing* is possible on nowadays mobile devices by the use of strong cryptography and short range communication. A set of various widespread available mobile devices has been used to derive a sufficient number of test runs to allow for sound predictions of the mandatory runtime and to identify potential pitfalls. Based on these findings, the *inner ratio* as well as the *outer ratio* have been introduced, motivated and derived for various settings. The first one expresses the ratio between operations of the same algorithm and keylength, in our case signature creation and verification. The scalability of a given algorithm with respect to the number of users can be expressed by the use of the inner ratio. Furthermore the suitability of the different signing schemas and communication patterns, that have been discussed, can be calculated by the use of the inner ratio. The later one, the outer ratio allows for a direct comparison between different key lengths of the same algorithm, such as RSA512 and RSA1024. Two more ratios, inter-algorithm and the inter-device ratio have been mentioned, but not consolidated.

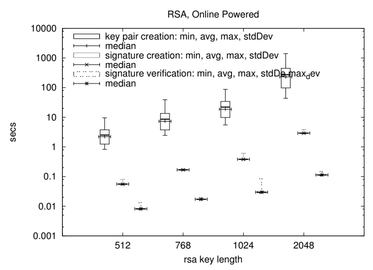
Besides the intuitive parallelized signing schema a more sophisticted one, the semi-sequential which makes use of the users knowledge, has been introduced and evaluated. It is optimal with respect to the number of signatures necessary. Different communication patters have been examined for both schemas. The existing tradeoff between the advantage of smaller computational costs on the one hand and on the other hand the problem of increased time consumption has been pointed out. Additionally, the mandatory communication has been examined. Although communication costs both, energy and time, only the timely aspect has been considered. The inner ratio has been identified as an important indicator, as it allows to rate the scalability depending on the number of users involved. It can also be used to rate the suitability of the different schedules as some of them rely stronger on sequential verifications. This is especially important if ECDSA with its high inner ratio is involved. It allows to determine the time needed for *incremental contract signing*, to choose the maximum possible security and to set up the best schedule under the given circumstands considering the interaction with the communication.

Although the emphasis was on the practical feasibility, the theoretical foundation have also been evaluated. An overview of digital signature algorithms has been provided. Then, available libraries have been rated according to their suitability for our project. Our main concerns were the free availability and of course a on-going development and improvement as well as open-source to be able to adapt the code to our needs if necessary. We also gave recommendations of the key length to be used within *incremental contract signing* to allow for secure and efficient fair offline trading.

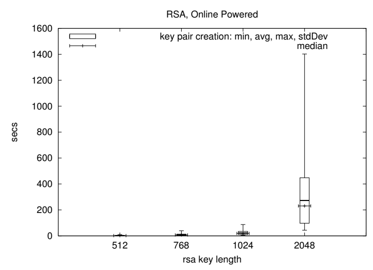
Appendix

A. Statistics

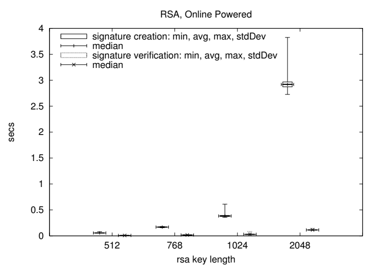
This part of the appendix will first provide the statistics of the two scenarios, which have been chosen in Section III as the two extremes to be evaluated, in Section A. For each of the scenarios the statistical data in form of minimal, maximal and average value are provided as error bars. Additional the median and the standard derivation is also integrated in the plots. In case of RSA, where the execution times of the cryptographic operations, key pair creation, signature creation and its verification differ that much, multiple plots are provided. A single plot to show all three of them and two separate ones for key pair creation and signature creation in combination with signature verification. The operations are abbreviated by letters, *C* for key pair creation, *S* for signature creation and *V* for verification. It can be seen that the standard deviation is small enough to allow for usage in productive systems. RSA key pair creation has due to its nature – checking large numbers to be prime – the highest standard deviation, which is of minor impact as key pairs can be created on fast devices or during times of no regular cell-phone usage. These findings can be observed in the following Sections B.1 and B.2 where the individual histograms for the two algorithms RSA and ECDSA are presented for each device and scenario.



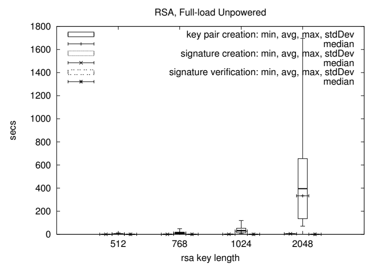
(a) RSA – CSV, Online Powered.



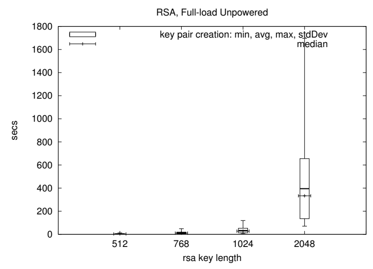
(b) RSA – Create.



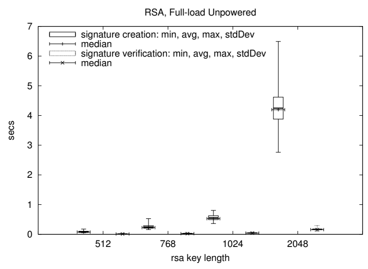
(c) RSA – SV, Online Powered.



(d) RSA – CSV, Full-load Unpowered.

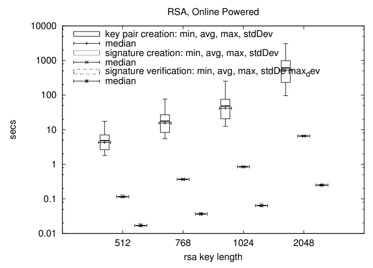


(e) RSA – C, Full-load Unpowered.

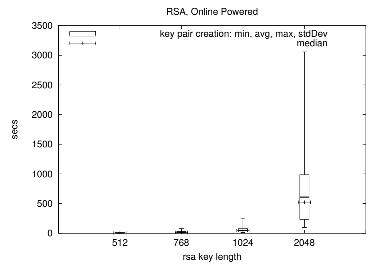


(f) RSA – SV, Full-load Unpowered.

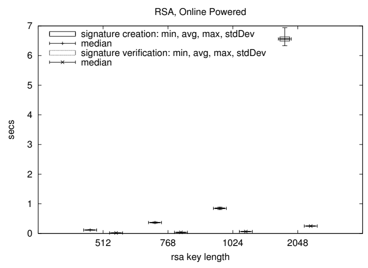
Fig. 20. M600i, RSA Error Bars, Online Powered and Full-load Unpowered.



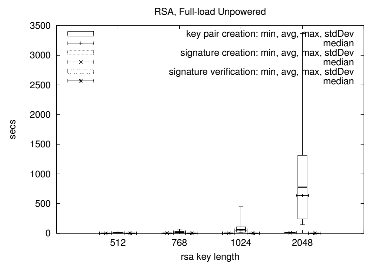
(a) RSA – CSV, Online Powered.



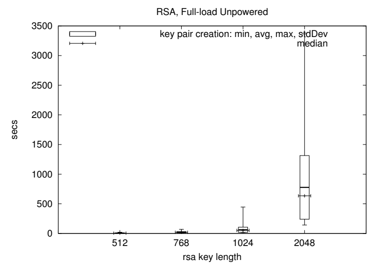
(b) RSA – C, Online Powered



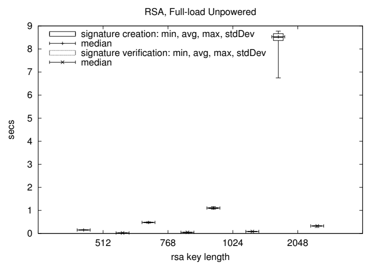
(c) RSA – SV, Online Powered.



(d) RSA – CSV, Full-load Unpowered.

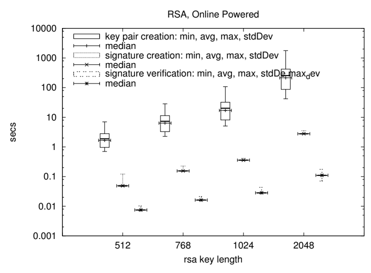


(e) RSA – C, Full-load Unpowered.

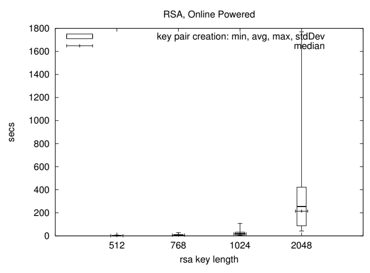


(f) RSA – SV, Full-load Unpowered.

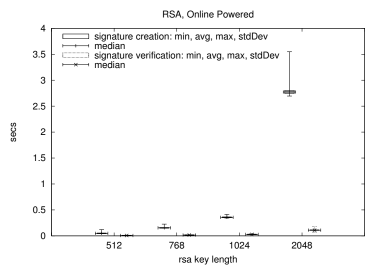
Fig. 21. 6300, RSA Error Bars, Online Powered and Full-load Unpowered.



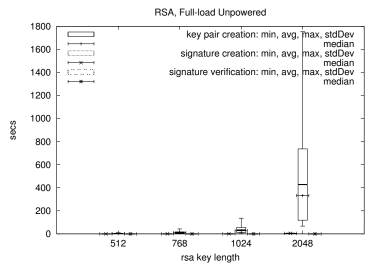
(a) RSA – CSV, Online Powered.



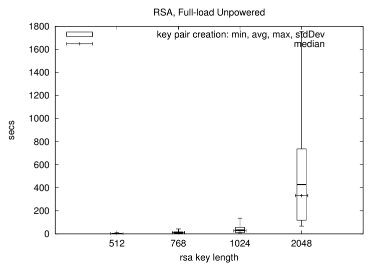
(b) RSA – C, Online Powered.



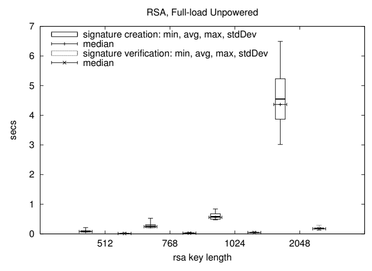
(c) RSA – SV, Online Powered.



(d) RSA – CSV, Online Powered.

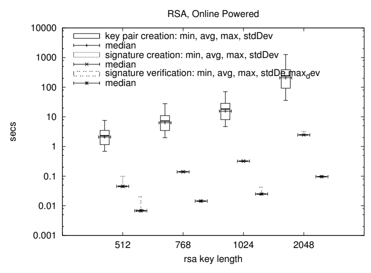


(e) RSA – C, Full-load Unpowered.

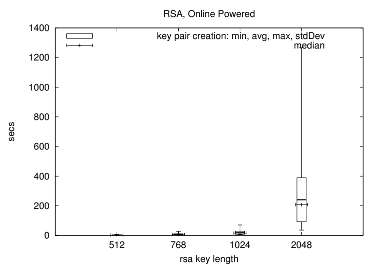


(f) RSA – SV, Full-load Unpowered.

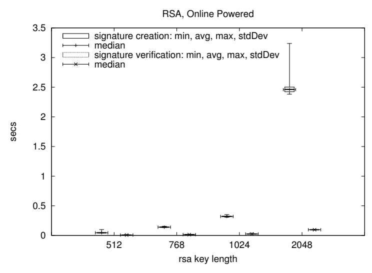
Fig. 22. 5800xm, RSA Error Bars, Online Powered and Full-load Unpowered.



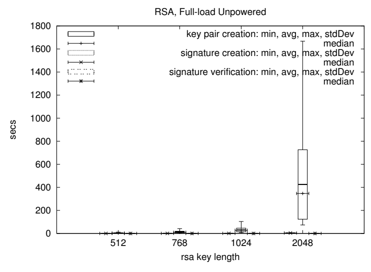
(a) RSA – CSV, Online Powered.



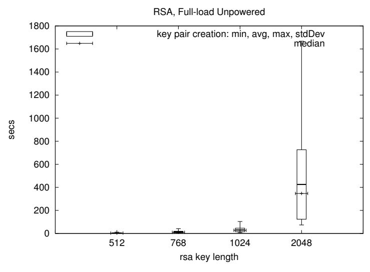
(b) RSA – C, Online Powered.



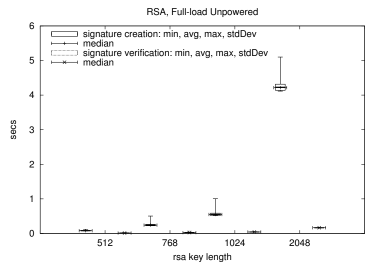
(c) RSA – SV, Online Powered.



(d) RSA – CSV, Full-load Unpowered.

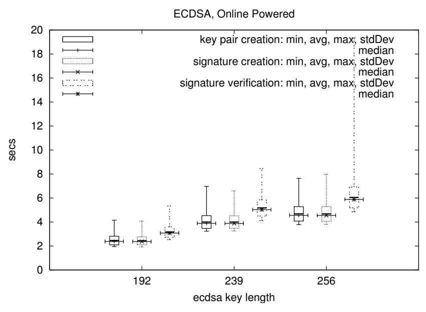


(e) RSA – C, Full-load Unpowered.

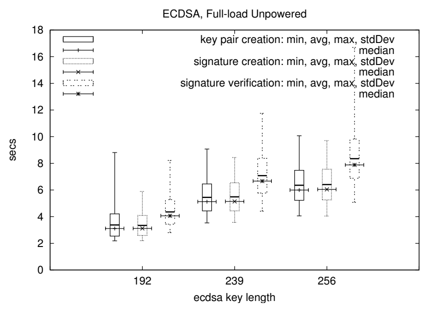


(f) RSA – SV, Full-load Unpowered.

Fig. 23. E50, RSA Error Bars, Online Powered and Full-load Unpowered.

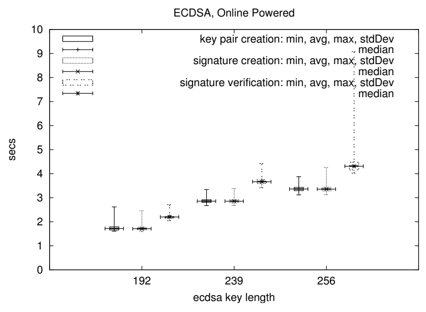


(a) ECDSA – CSV, Online Powered.

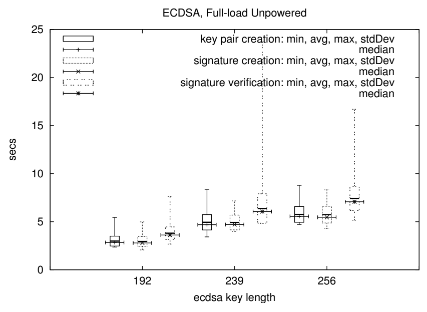


(b) ECDSA – CSV, Full-load Unpowered.

Fig. 24. M600i, ECDSA Error Bars, Online Powered and Full-load Unpowered.

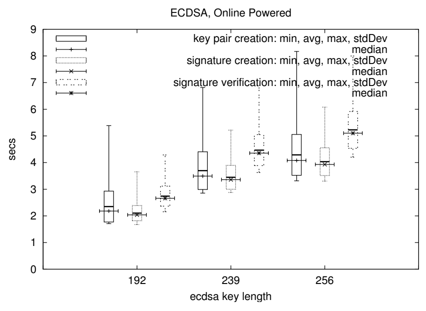


(a) ECDSA – CSV, Online Powered.

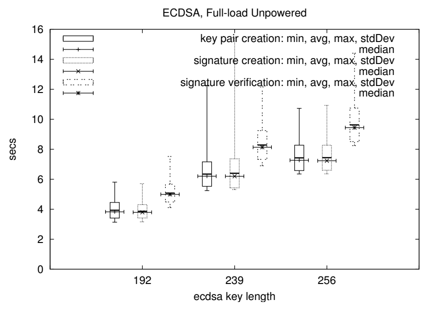


(b) ECDSA – CSV, Full-load Unpowered.

Fig. 25. 5800xm, ECDSA Error Bars, Online Powered and Full-load Unpowered.



(a) ECDSA – CSV, Online Powered.



(b) ECDSA – CSV, Full-load Unpowered.

Fig. 26. E50, ECDSA Error Bars, Online Powered and Full-load Unpowered.

B. Histograms

1) RSA Histograms: The figures of the following section contain the histograms for the RSA algorithm for all four considered key lengths: 512, 768, 1024 and 2048. They are ordered by device (M600i, 6300, 5800xm, E50, S65, K750i) in the first place, then by setting (online powered, offline powered, online unpowered, full-load powered, full-load unpowered). Within the figures each operation (creation of the key pair, creation of the signature and signature verification) has its own column. The key length increases from top to bottom (512, 768, 1024, 2048), while the last row provides plots with all key length inside a single plot. These histograms have been derived from the entire set of measurements.

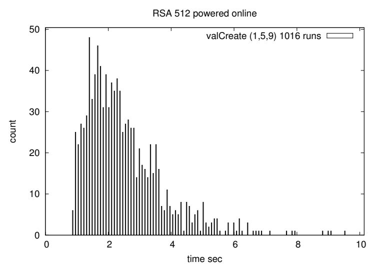
The following figures contain the histograms for the respective devices:

- *M600i.* Figure 27(a) to 31(o),
- *6300.* Figure 32(a) to 36(o),
- *5800XM.* Figure 37(a) to 40(o),
- *E50.* Figure 42(a) to 46(o),
- *S65.* Figure 47(a) to 47(o),
- *K750i.* Figure 48(a) to 48(o).

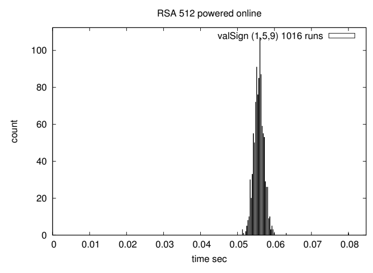
2) ECDSA Histograms: The figures of the following section contain the histograms for the ECDSA algorithm for three keylengths: 192, 239, 256. They are ordered by device (M600i, 5800XM, E50, S65) in the first place, then by setting (online powered, offline powered, online unpowered, full-load powered, full-load unpowered). Within the figures each operation (creation of the key pair, creation of the signature and signature verification) has its own column. The key length increases from top to bottom (192, 239, 256), while the last row provides plots with all key length inside a single plot. These histograms are derived from the entire set of measurements.

The following figures contain the histograms for the respective devices:

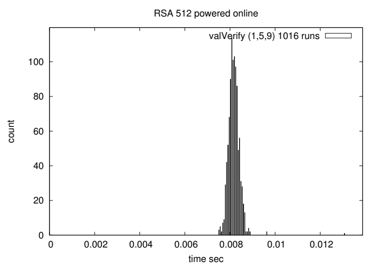
- *M600i.* Figure 49(a) to 53(l)
- *5800XM.* Figure 54(a) to 58(l)
- *E50.* Figure 59(a) to 63(l)
- *S65.* Figure 64(a) to 64(l).



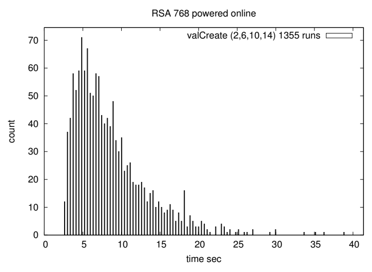
(a) RSA512, Key Pair Creation.



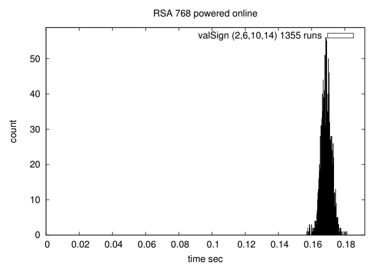
(b) RSA512, Signature Creation.



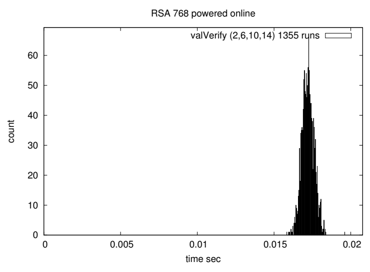
(c) RSA512, Signature Verification.



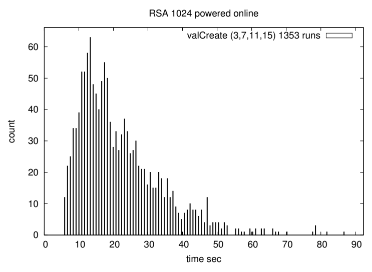
(d) RSA768, Key Pair Creation.



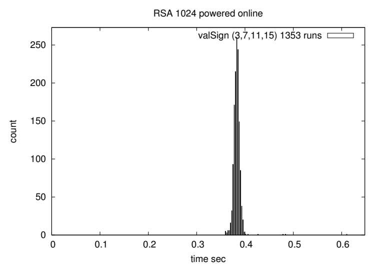
(e) RSA768, Signature Creation.



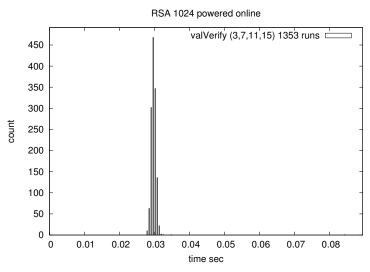
(f) RSA768, Signature Verification.



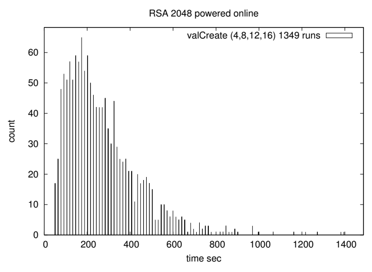
(g) RSA1024, Key Pair Creation.



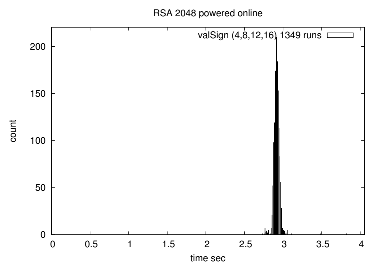
(h) RSA1024, Signature Creation.



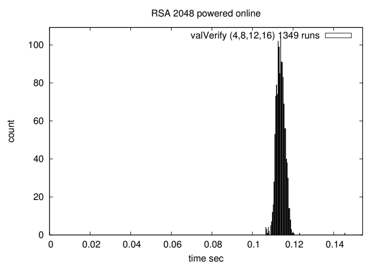
(i) RSA1024, Signature Verification.



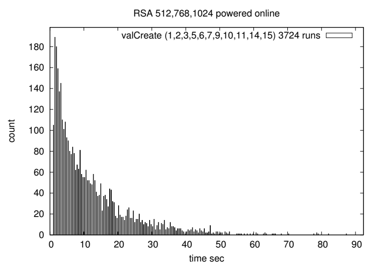
(j) RSA2048, Key Pair Creation.



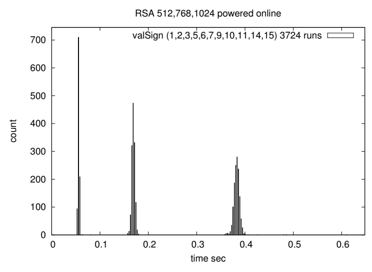
(k) RSA2048, Signature Creation.



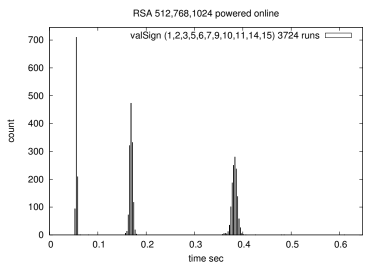
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

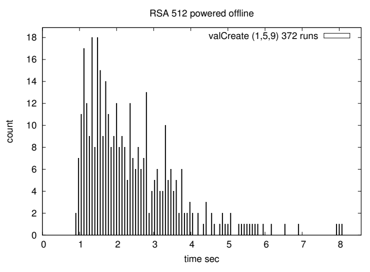


(n) RSA -1024, Signature Creation.

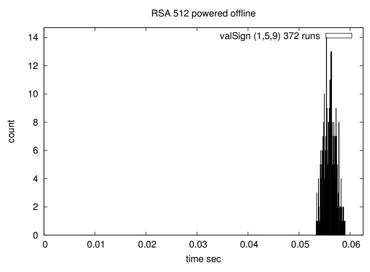


(o) RSA -1024, Signature Verification.

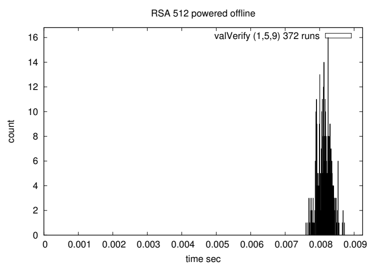
Fig. 27. M600i, Online Powered, RSA 512, 768, 1024, 2048.



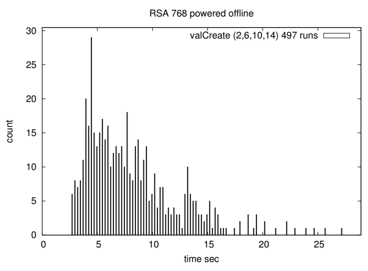
(a) RSA512, Key Pair Creation.



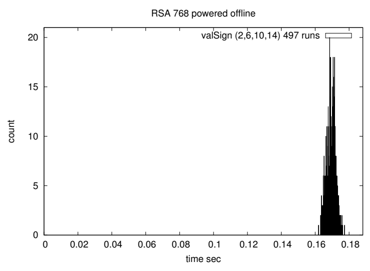
(b) RSA512, Signature Creation.



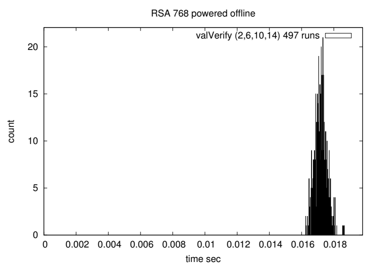
(c) RSA512, Signature Verification.



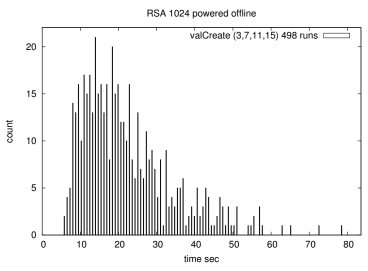
(d) RSA768, Key Pair Creation.



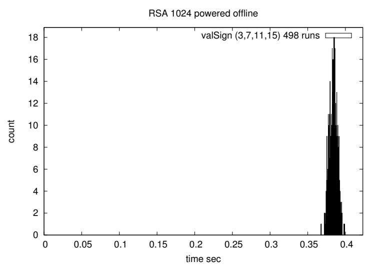
(e) RSA768, Signature Creation.



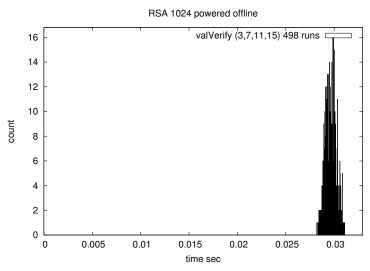
(f) RSA768, Signature Verification.



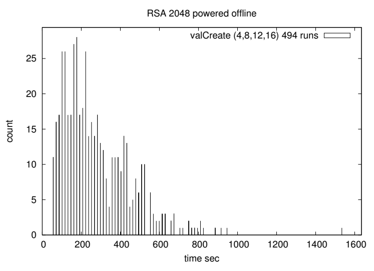
(g) RSA1024, Key Pair Creation.



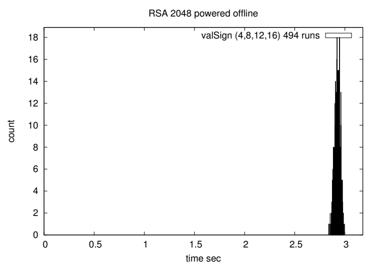
(h) RSA1024, Signature Creation.



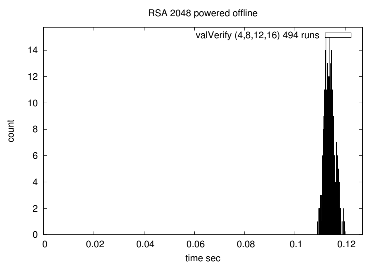
(i) RSA1024, Signature Verification.



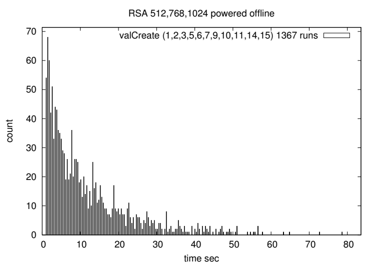
(j) RSA2048, Key Pair Creation.



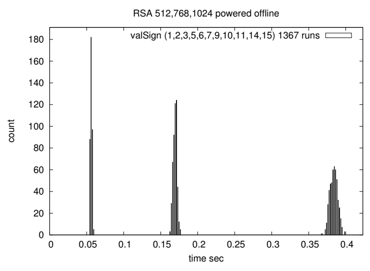
(k) RSA2048, Signature Creation.



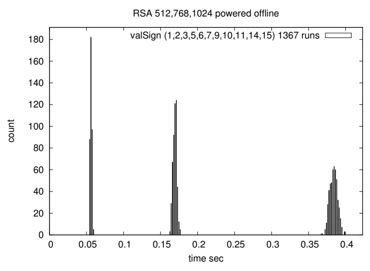
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

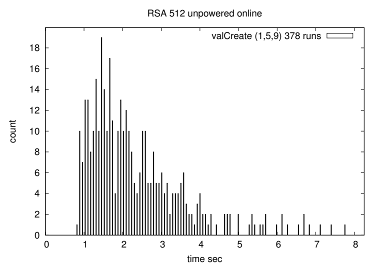


(n) RSA -1024, Signature Creation.

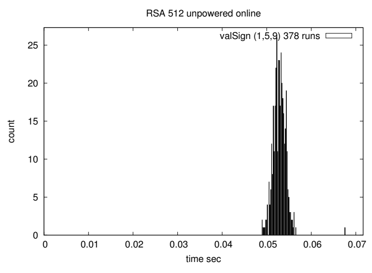


(o) RSA -1024, Signature Verification.

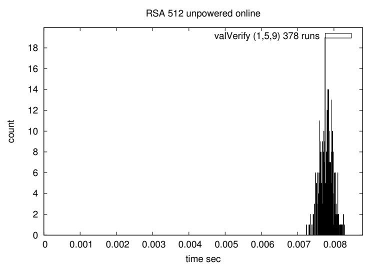
Fig. 28. M600i, Offline Powered, RSA 512, 768, 1024, 2048.



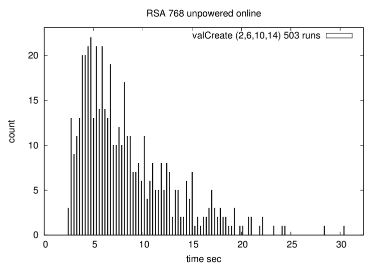
(a) RSA512, Key Pair Creation.



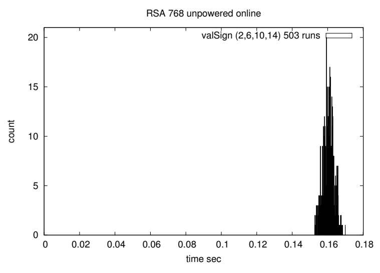
(b) RSA512, Signature Creation.



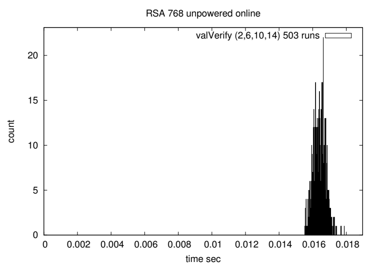
(c) RSA512, Signature Verification.



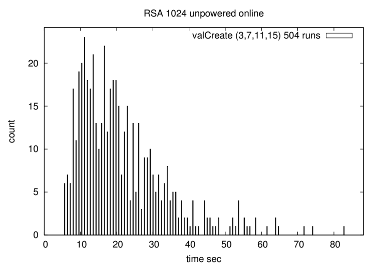
(d) RSA768, Key Pair Creation.



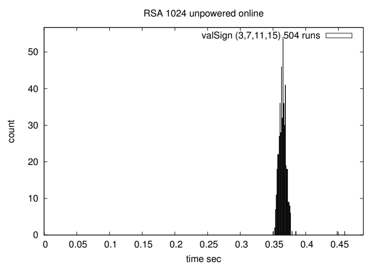
(e) RSA768, Signature Creation.



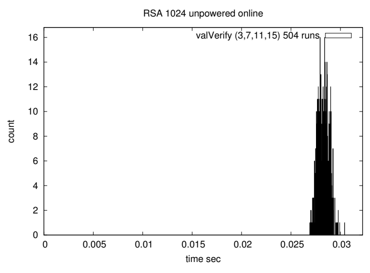
(f) RSA768, Signature Verification.



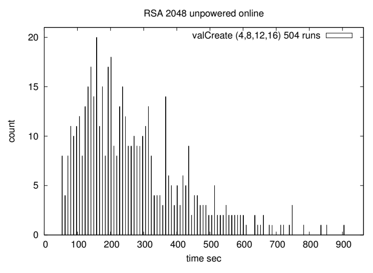
(g) RSA1024, Key Pair Creation.



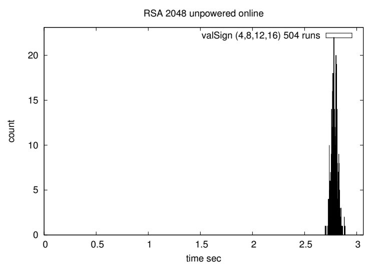
(h) RSA1024, Signature Creation.



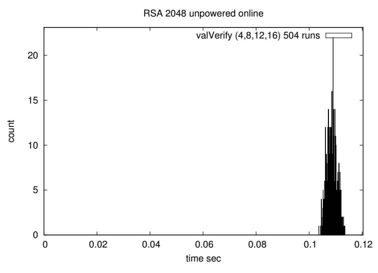
(i) RSA1024, Signature Verification.



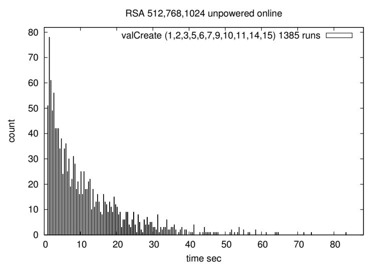
(j) RSA2048, Key Pair Creation.



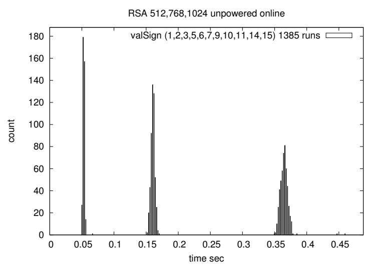
(k) RSA2048, Signature Creation.



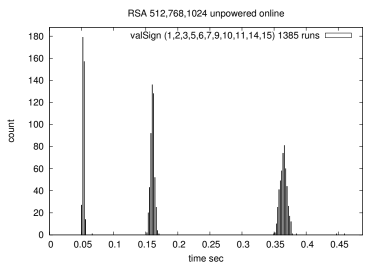
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

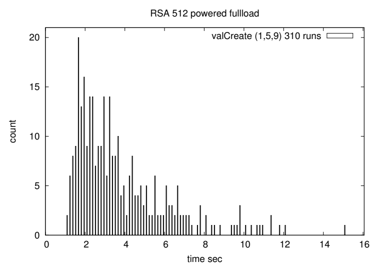


(n) RSA -1024, Signature Creation.

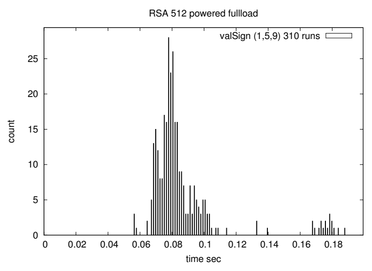


(o) RSA -1024, Signature Verification.

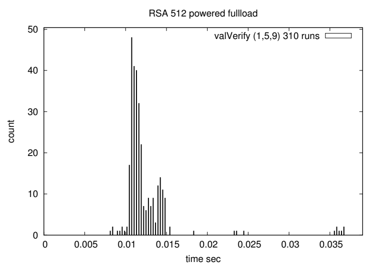
Fig. 29. M600i, Online Unpowered, RSA 512, 768, 1024, 2048.



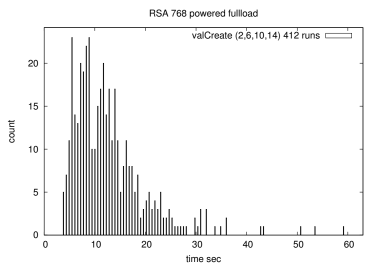
(a) RSA512, Key Pair Creation.



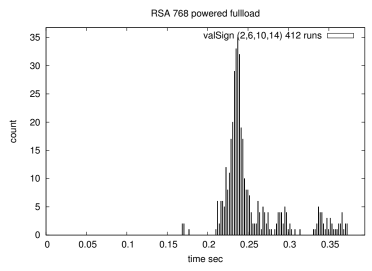
(b) RSA512, Signature Creation.



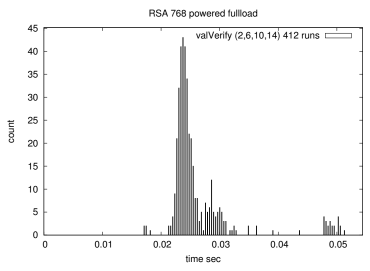
(c) RSA512, Signature Verification.



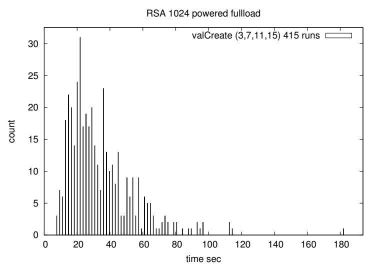
(d) RSA768, Key Pair Creation.



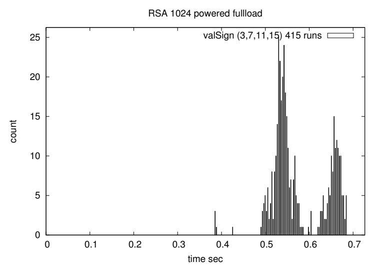
(e) RSA768, Signature Creation.



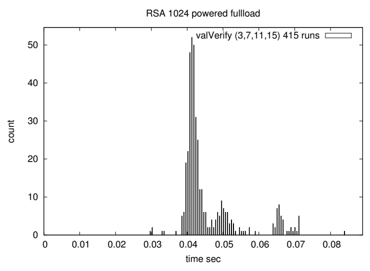
(f) RSA768, Signature Verification.



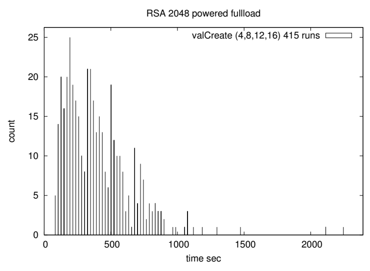
(g) RSA1024, Key Pair Creation.



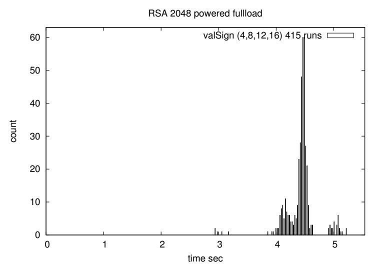
(h) RSA1024, Signature Creation.



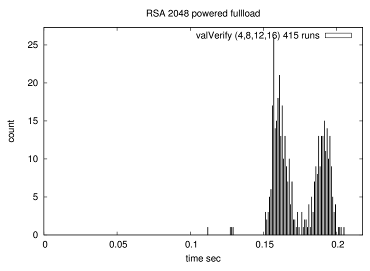
(i) RSA1024, Signature Verification.



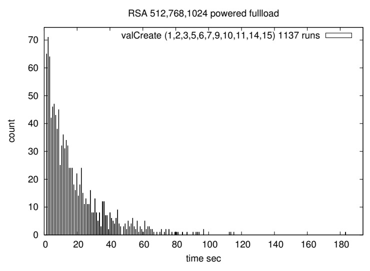
(j) RSA2048, Key Pair Creation.



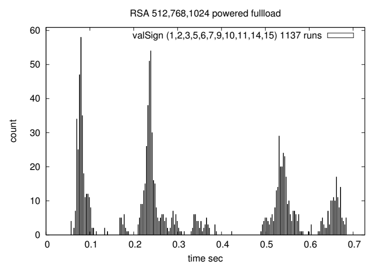
(k) RSA2048, Signature Creation.



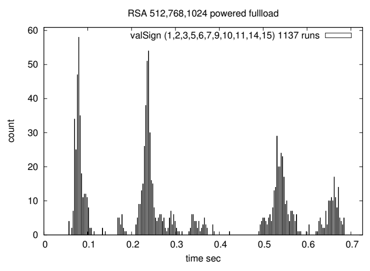
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

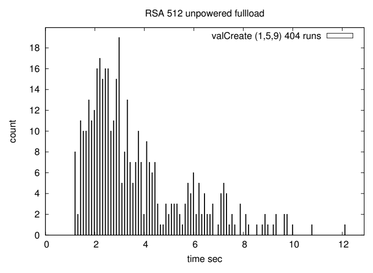


(n) RSA -1024, Signature Creation.

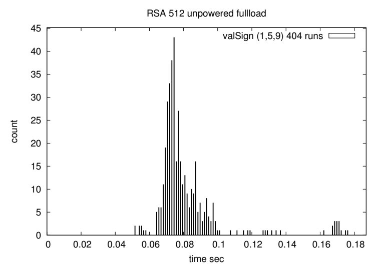


(o) RSA -1024, Signature Verification.

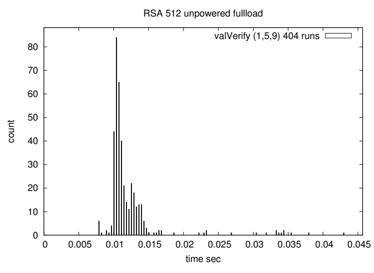
Fig. 30. M600i, Full-load Powered, RSA 512, 768, 1024, 2048.



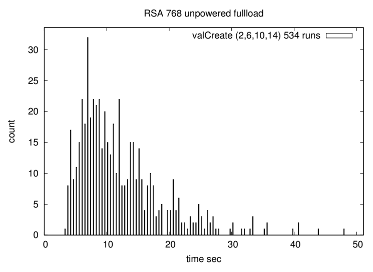
(a) RSA512, Key Pair Creation.



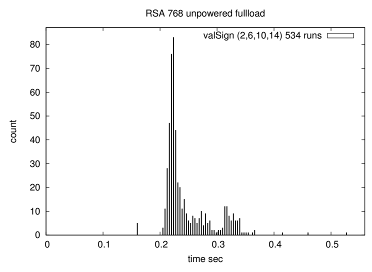
(b) RSA512, Signature Creation.



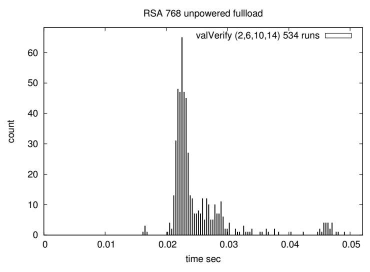
(c) RSA512, Signature Verification.



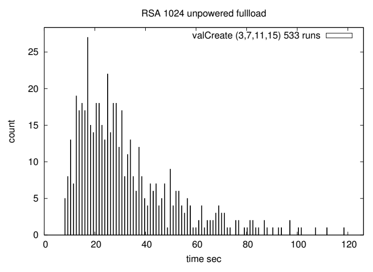
(d) RSA768, Key Pair Creation.



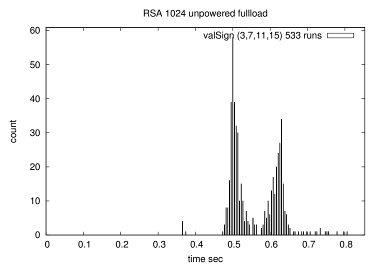
(e) RSA768, Signature Creation.



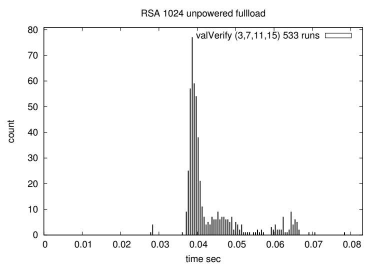
(f) RSA768, Signature Verification.



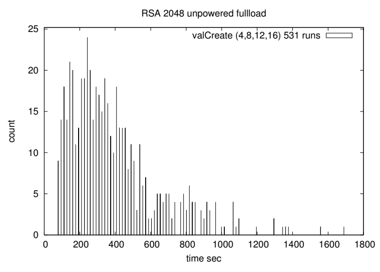
(g) RSA1024, Key Pair Creation.



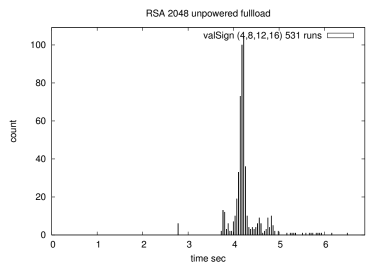
(h) RSA1024, Signature Creation.



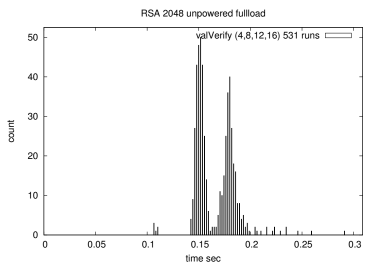
(i) RSA1024, Signature Verification.



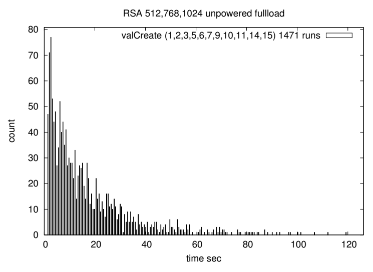
(j) RSA2048, Key Pair Creation.



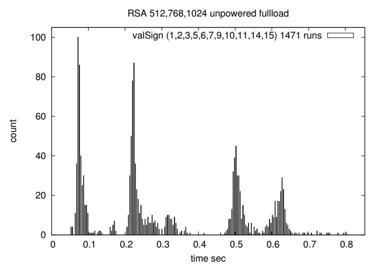
(k) RSA2048, Signature Creation.



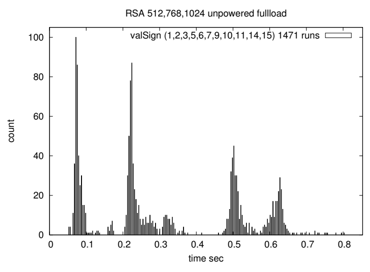
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

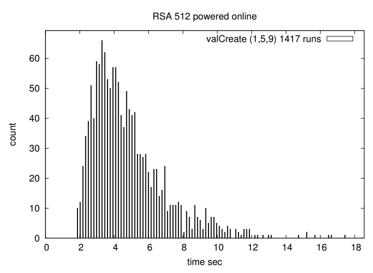


(n) RSA -1024, Signature Creation.

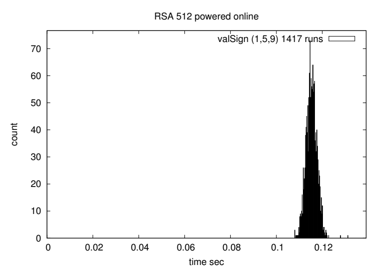


(o) RSA -1024, Signature Verification.

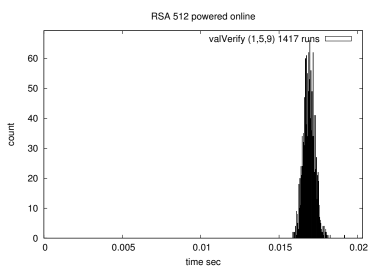
Fig. 31. M600i, Full-load Unpowered, RSA 512, 768, 1024, 2048.



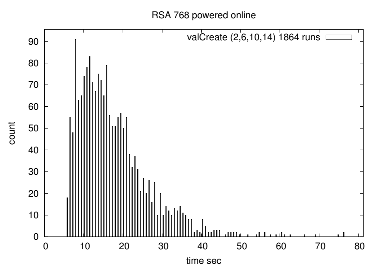
(a) RSA512, Key Pair Creation.



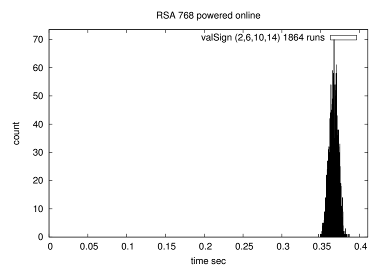
(b) RSA512, Signature Creation.



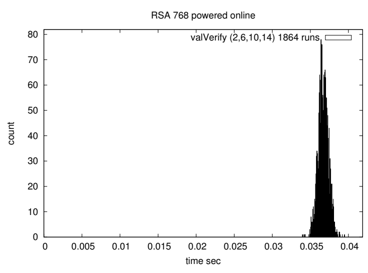
(c) RSA512, Signature Verification.



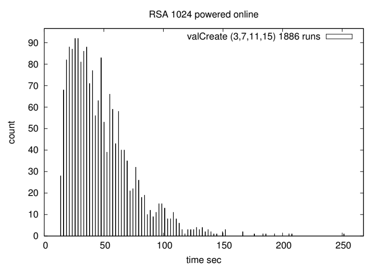
(d) RSA768, Key Pair Creation.



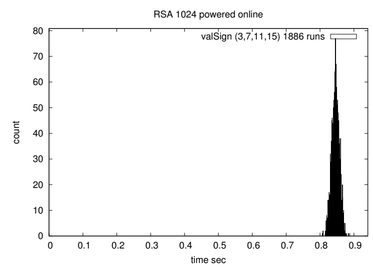
(e) RSA768, Signature Creation.



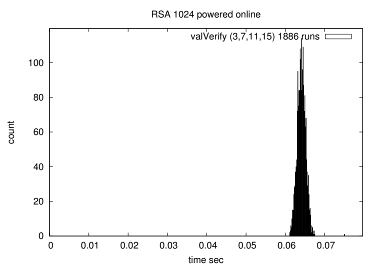
(f) RSA768, Signature Verification.



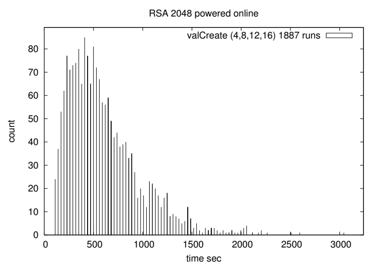
(g) RSA1024, Key Pair Creation.



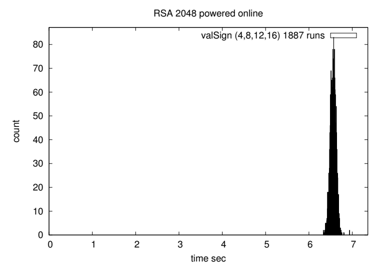
(h) RSA1024, Signature Creation.



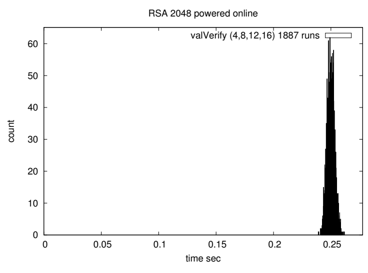
(i) RSA1024, Signature Verification.



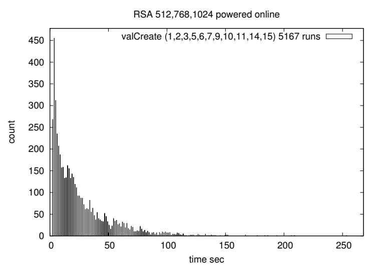
(j) RSA2048, Key Pair Creation.



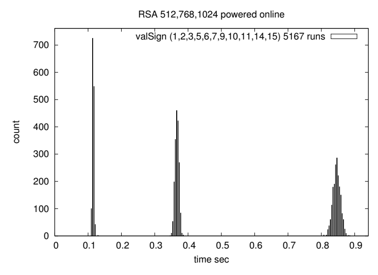
(k) RSA2048, Signature Creation.



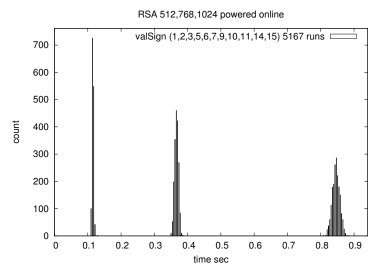
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

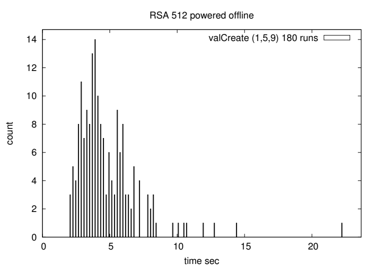


(n) RSA -1024, Signature Creation.

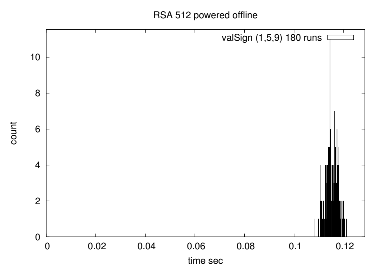


(o) RSA -1024, Signature Verification.

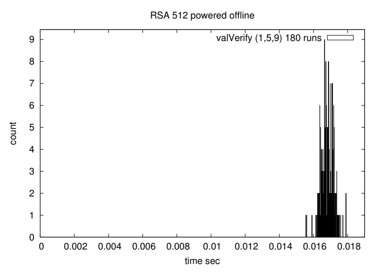
Fig. 32. 6300, Online Powered, RSA 512, 768, 1024, 2048.



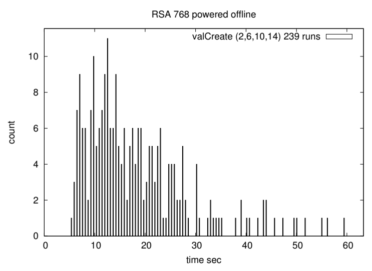
(a) RSA512, Key Pair Creation.



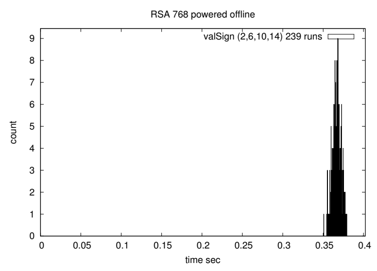
(b) RSA512, Signature Creation.



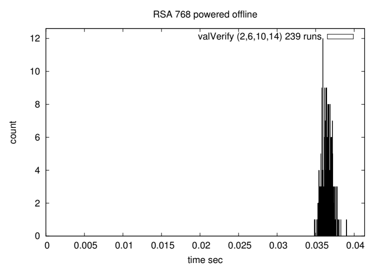
(c) RSA512, Signature Verification.



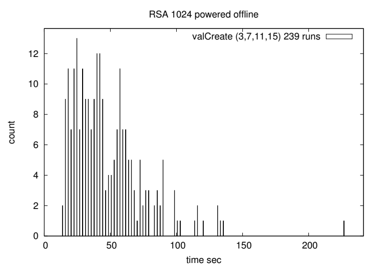
(d) RSA768, Key Pair Creation.



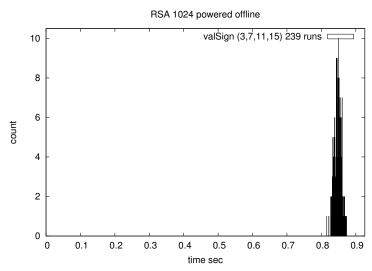
(e) RSA768, Signature Creation.



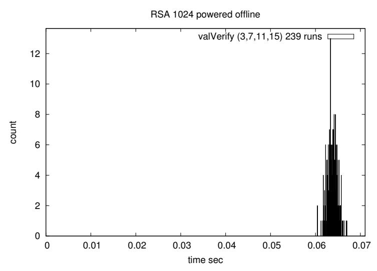
(f) RSA768, Signature Verification.



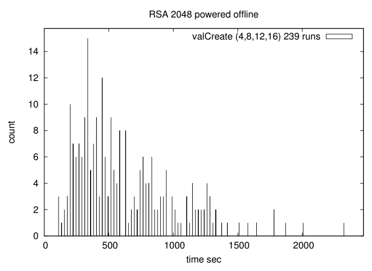
(g) RSA1024, Key Pair Creation.



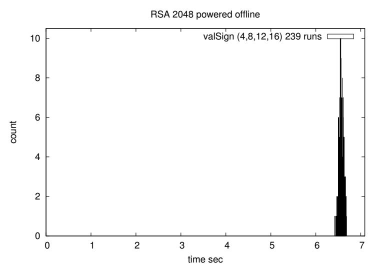
(h) RSA1024, Signature Creation.



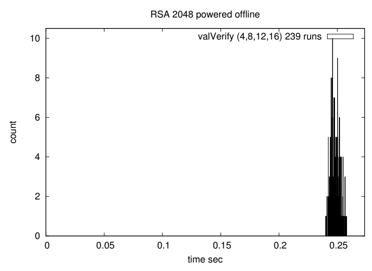
(i) RSA1024, Signature Verification.



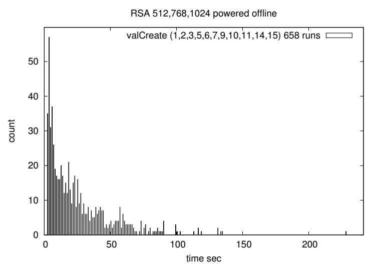
(j) RSA2048, Key Pair Creation.



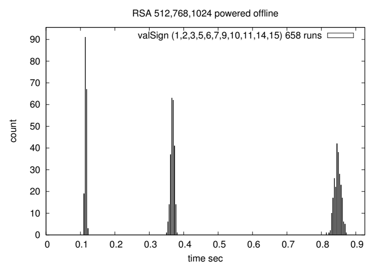
(k) RSA2048, Signature Creation.



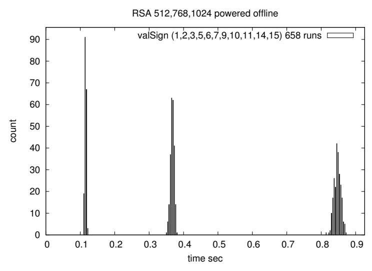
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

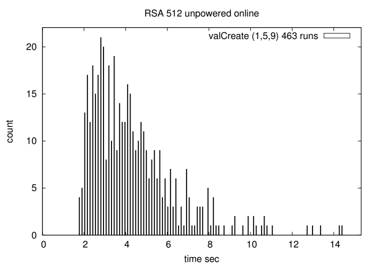


(n) RSA -1024, Signature Creation.

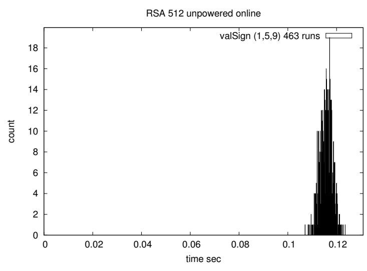


(o) RSA -1024, Signature Verification.

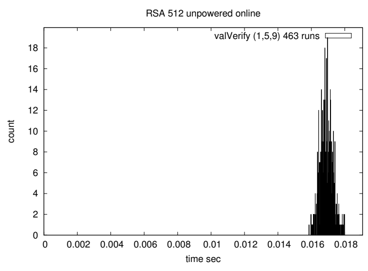
Fig. 33. 6300, Offline Powered, RSA 512, 768, 1024, 2048.



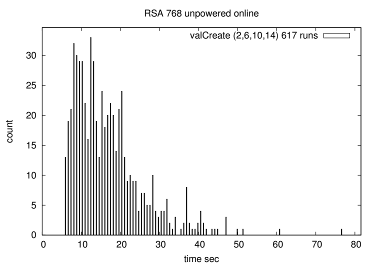
(a) RSA512, Key Pair Creation.



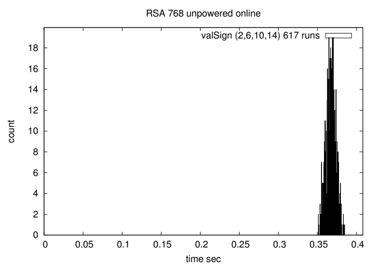
(b) RSA512, Signature Creation.



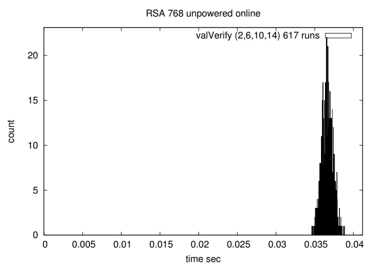
(c) RSA512, Signature Verification.



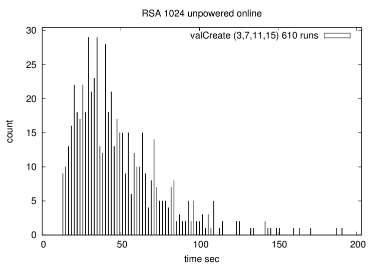
(d) RSA768, Key Pair Creation.



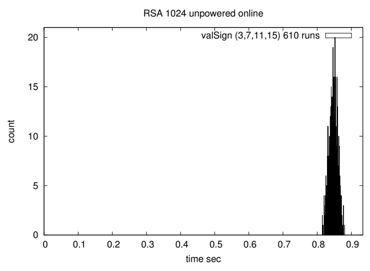
(e) RSA768, Signature Creation.



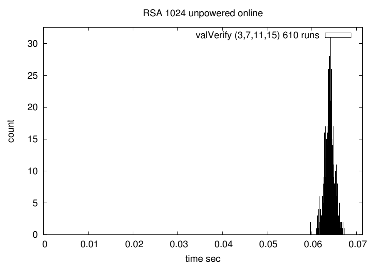
(f) RSA768, Signature Verification.



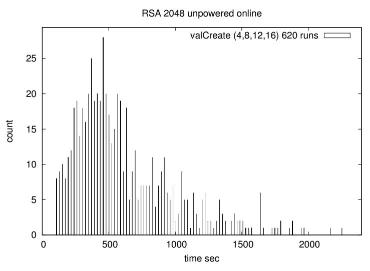
(g) RSA1024, Key Pair Creation.



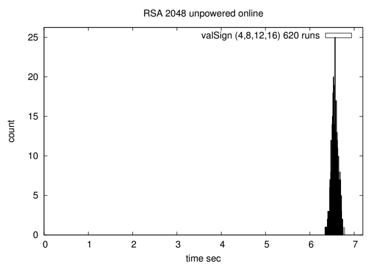
(h) RSA1024, Signature Creation.



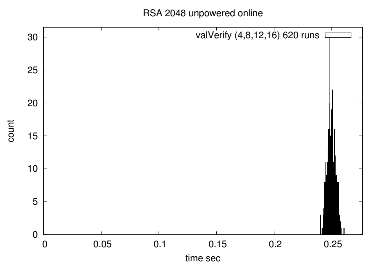
(i) RSA1024, Signature Verification.



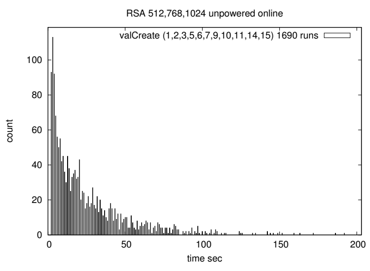
(j) RSA2048, Key Pair Creation.



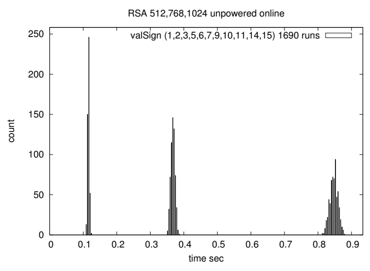
(k) RSA2048, Signature Creation.



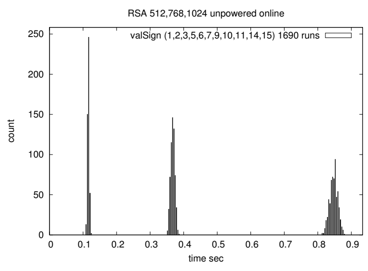
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

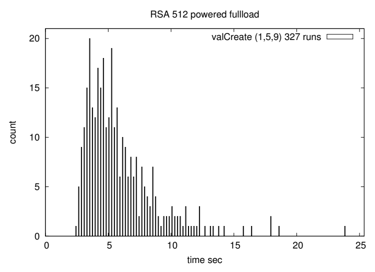


(n) RSA -1024, Signature Creation.

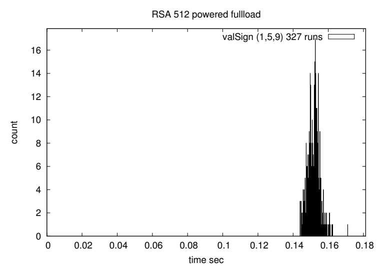


(o) RSA -1024, Signature Verification.

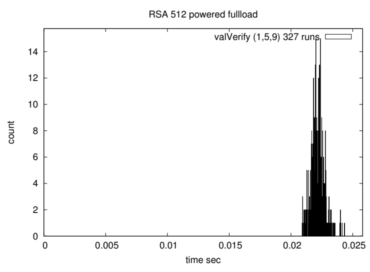
Fig. 34. 6300, Online Unpowered, RSA 512, 768, 1024, 2048.



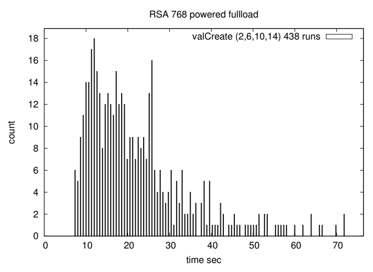
(a) RSA512, Key Pair Creation.



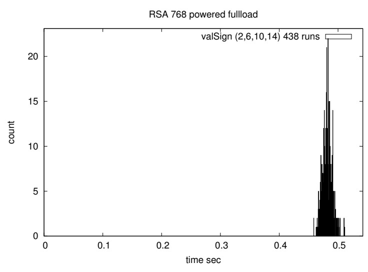
(b) RSA512, Signature Creation.



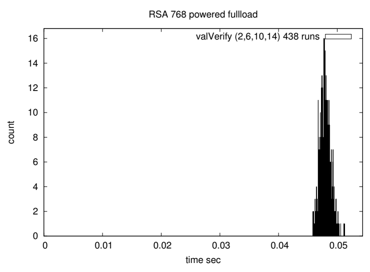
(c) RSA512, Signature Verification.



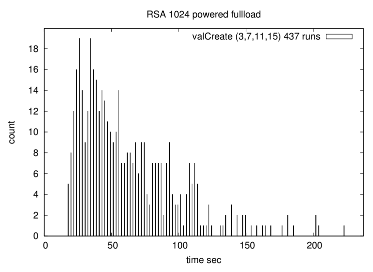
(d) RSA768, Key Pair Creation.



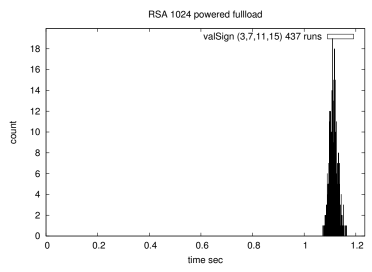
(e) RSA768, Signature Creation.



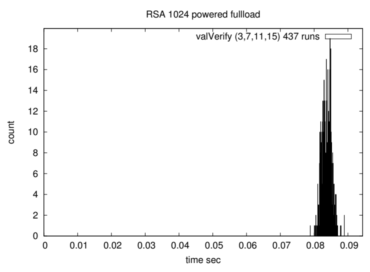
(f) RSA768, Signature Verification.



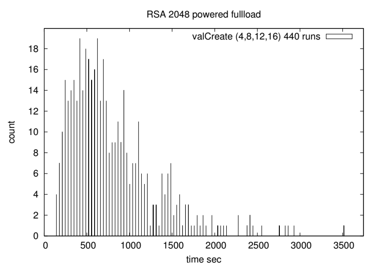
(g) RSA1024, Key Pair Creation.



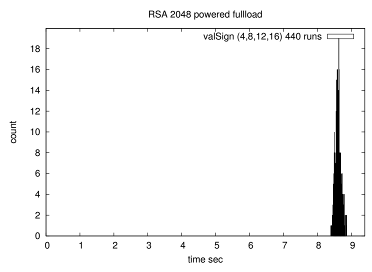
(h) RSA1024, Signature Creation.



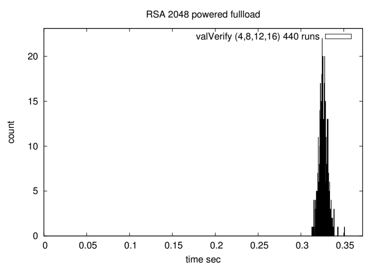
(i) RSA1024, Signature Verification.



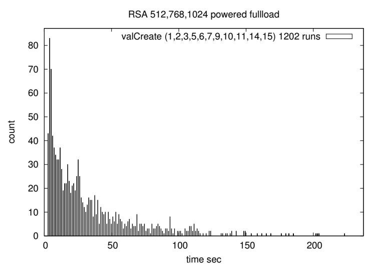
(j) RSA2048, Key Pair Creation.



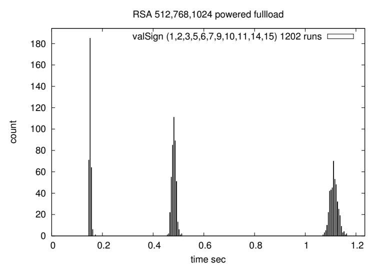
(k) RSA2048, Signature Creation.



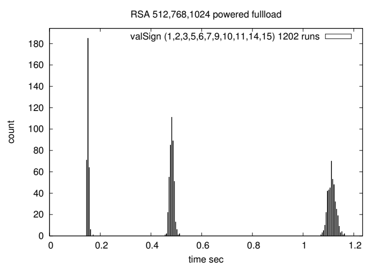
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

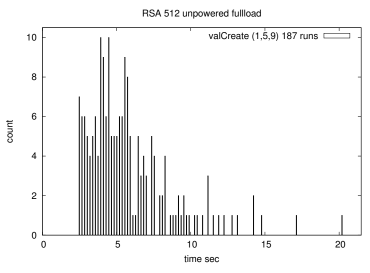


(n) RSA -1024, Signature Creation.

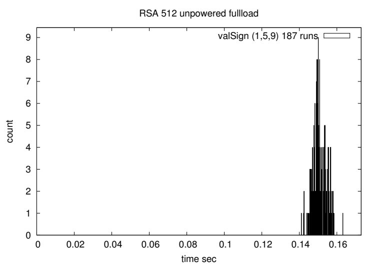


(o) RSA -1024, Signature Verification.

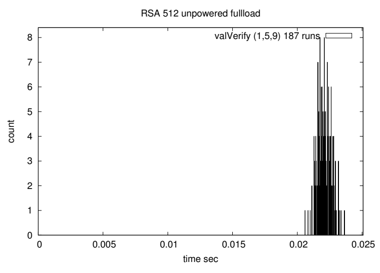
Fig. 35. 6300, Fullload Powered, RSA 512, 768, 1024, 2048.



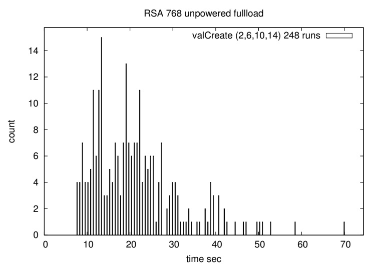
(a) RSA512, Key Pair Creation.



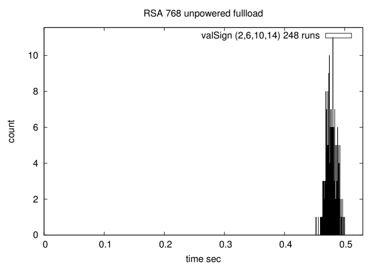
(b) RSA512, Signature Creation.



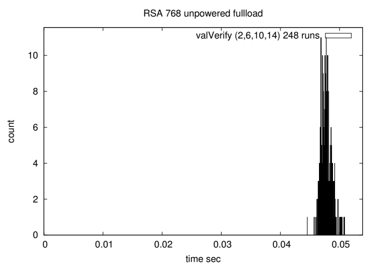
(c) RSA512, Signature Verification.



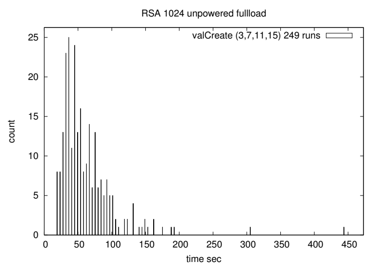
(d) RSA768, Key Pair Creation.



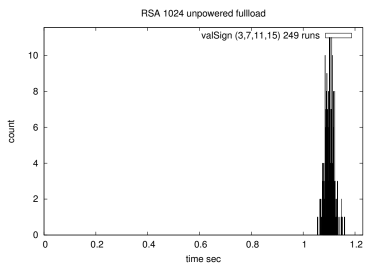
(e) RSA768, Signature Creation.



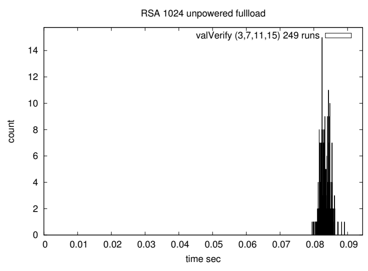
(f) RSA768, Signature Verification.



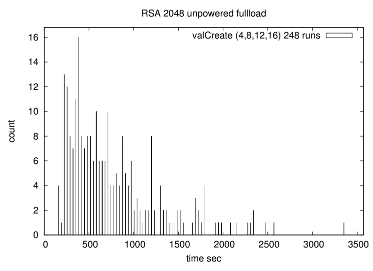
(g) RSA1024, Key Pair Creation.



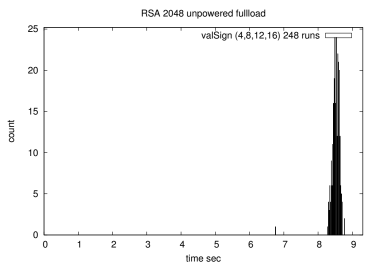
(h) RSA1024, Signature Creation.



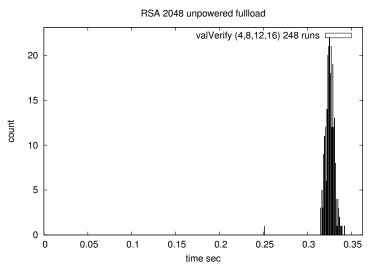
(i) RSA1024, Signature Verification.



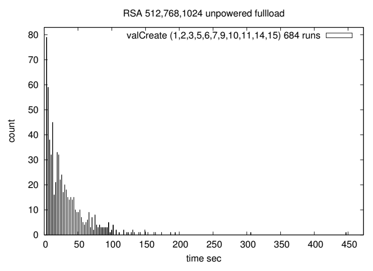
(j) RSA2048, Key Pair Creation.



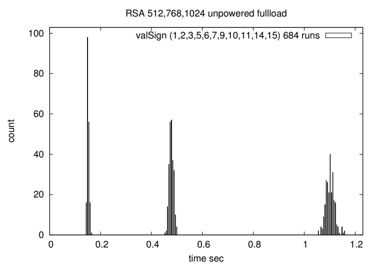
(k) RSA2048, Signature Creation.



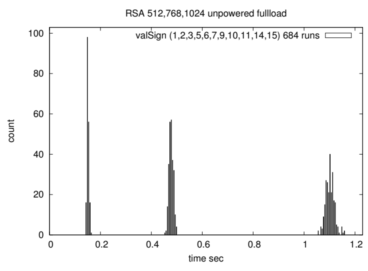
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

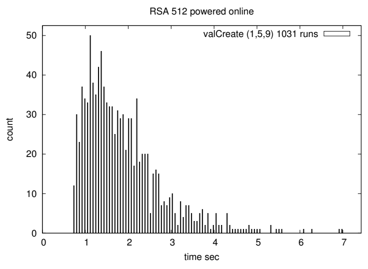


(n) RSA -1024, Signature Creation.

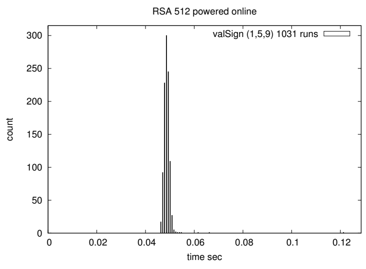


(o) RSA -1024, Signature Verification.

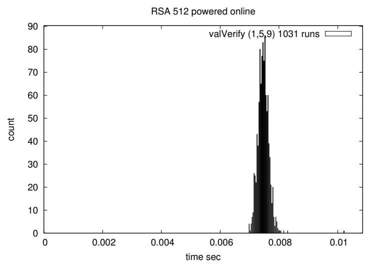
Fig. 36. 6300, Full-load Unpowered, RSA 512, 768, 1024, 2048.



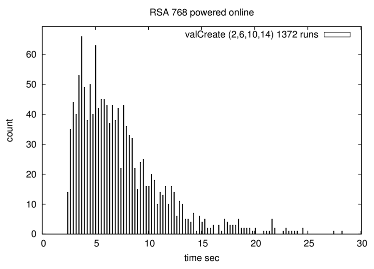
(a) RSA512, Key Pair Creation.



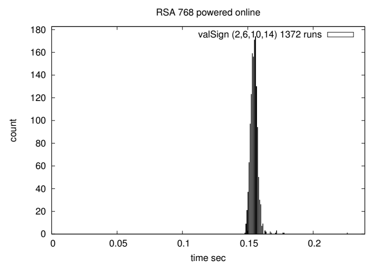
(b) RSA512, Signature Creation.



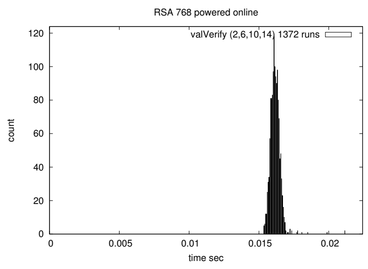
(c) RSA512, Signature Verification.



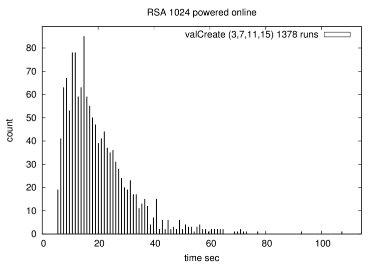
(d) RSA768, Key Pair Creation.



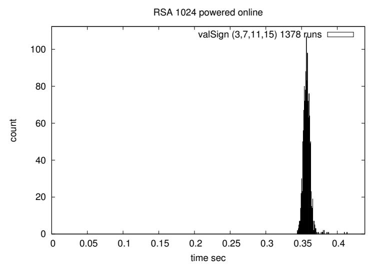
(e) RSA768, Signature Creation.



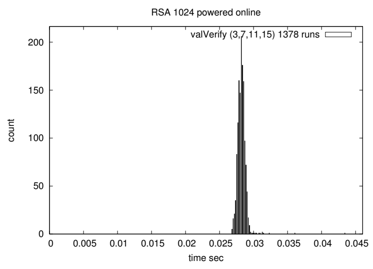
(f) RSA768, Signature Verification.



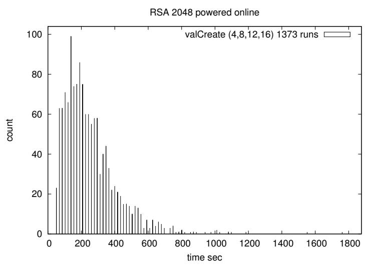
(g) RSA1024, Key Pair Creation.



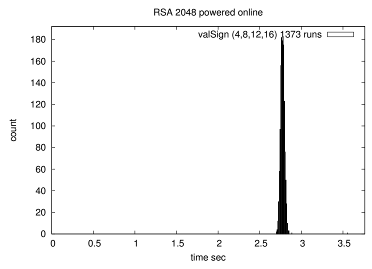
(h) RSA1024, Signature Creation.



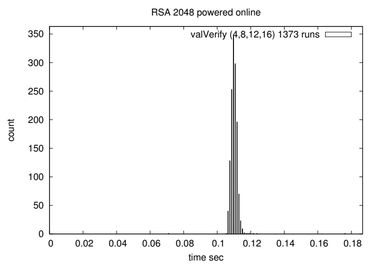
(i) RSA1024, Signature Verification.



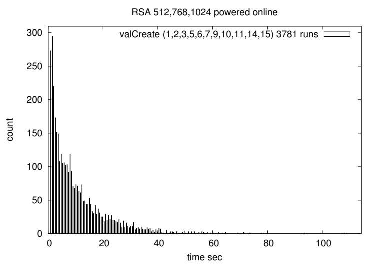
(j) RSA2048, Key Pair Creation.



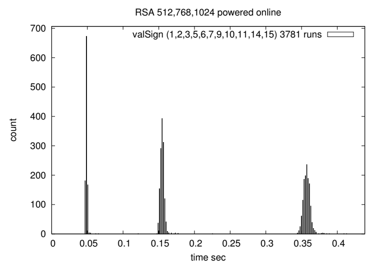
(k) RSA2048, Signature Creation.



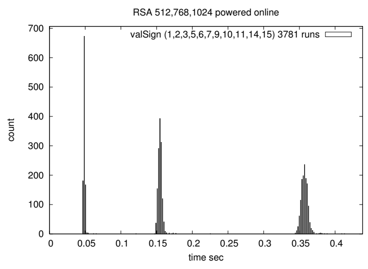
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

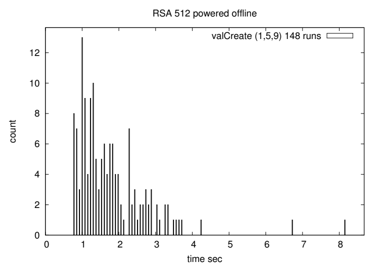


(n) RSA -1024, Signature Creation.

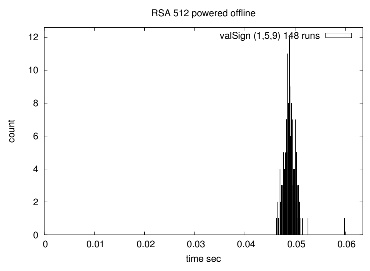


(o) RSA -1024, Signature Verification.

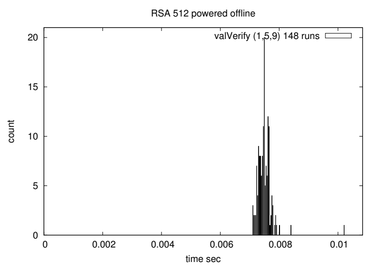
Fig. 37. 5800xm, Online Powered, RSA 512, 768, 1024, 2048.



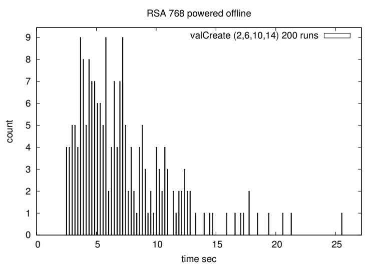
(a) RSA512, Key Pair Creation.



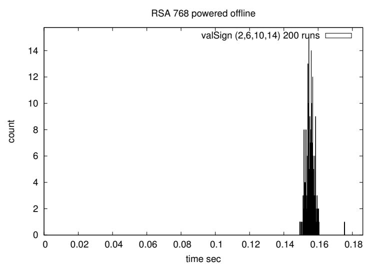
(b) RSA512, Signature Creation.



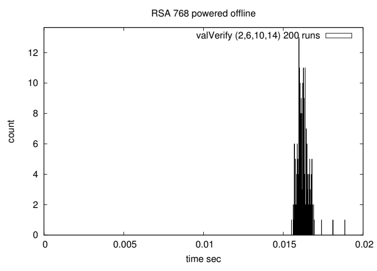
(c) RSA512, Signature Verification.



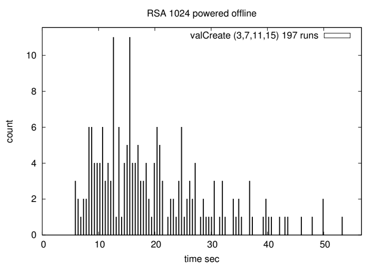
(d) RSA768, Key Pair Creation.



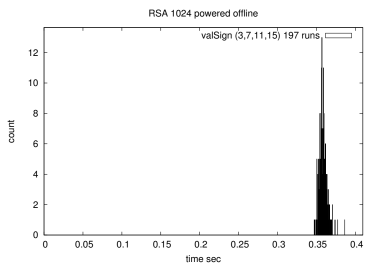
(e) RSA768, Signature Creation.



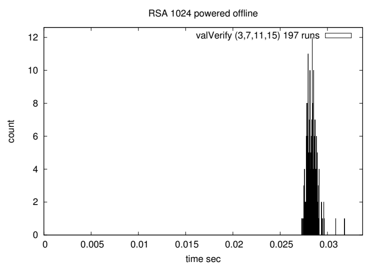
(f) RSA768, Signature Verification.



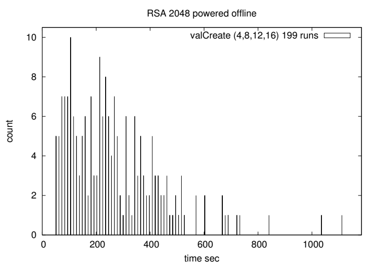
(g) RSA1024, Key Pair Creation.



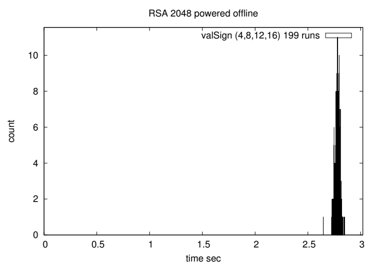
(h) RSA1024, Signature Creation.



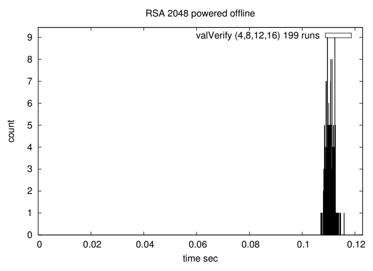
(i) RSA1024, Signature Verification.



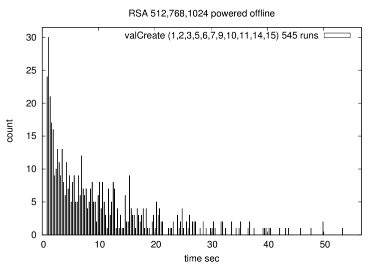
(j) RSA2048, Key Pair Creation.



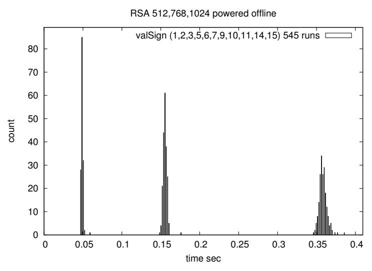
(k) RSA2048, Signature Creation.



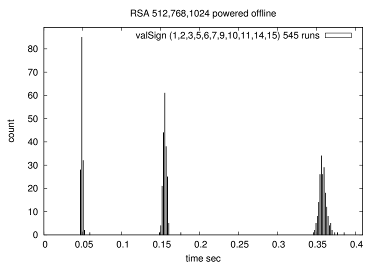
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

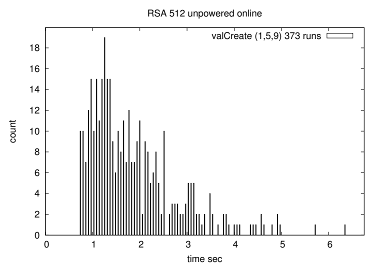


(n) RSA -1024, Signature Creation.

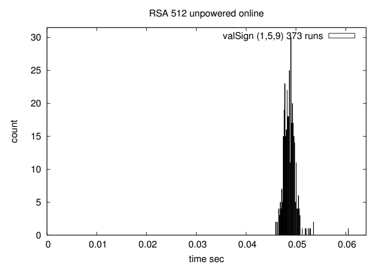


(o) RSA -1024, Signature Verification.

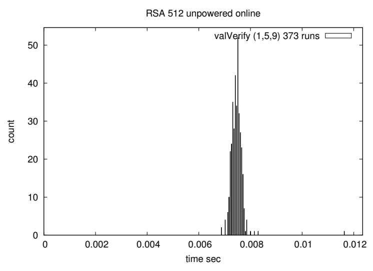
Fig. 38. 5800xm, Offline Powered, RSA 512, 768, 1024, 2048.



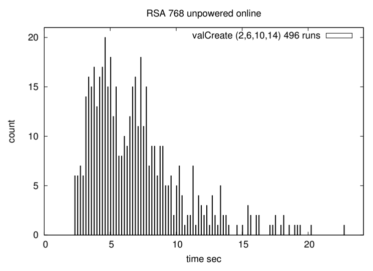
(a) RSA512, Key Pair Creation.



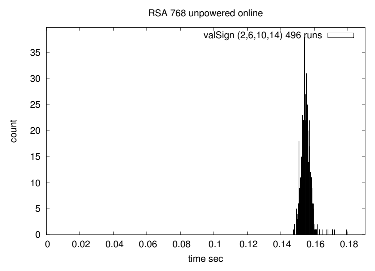
(b) RSA512, Signature Creation.



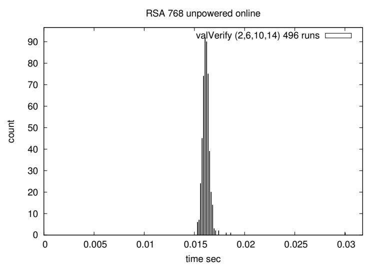
(c) RSA512, Signature Verification.



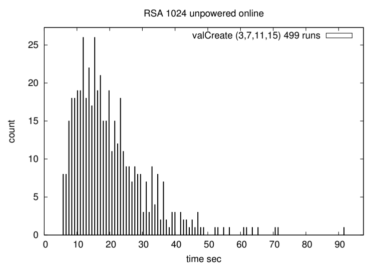
(d) RSA768, Key Pair Creation.



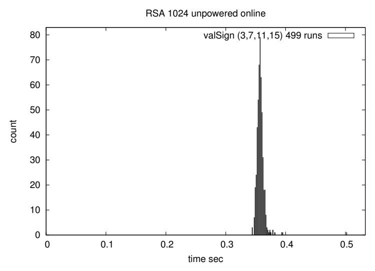
(e) RSA768, Signature Creation.



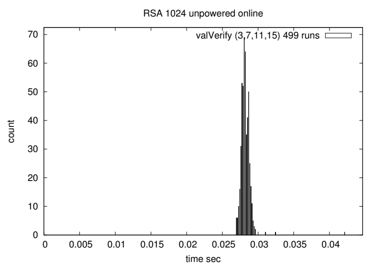
(f) RSA768, Signature Verification.



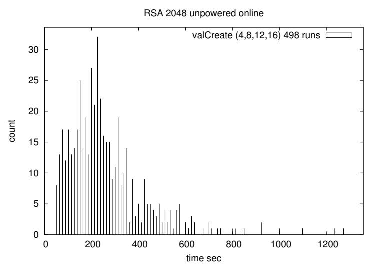
(g) RSA1024, Key Pair Creation.



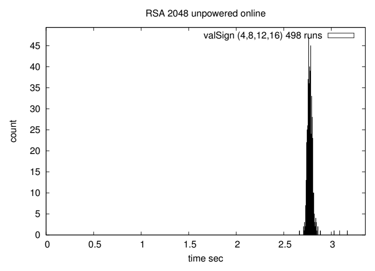
(h) RSA1024, Signature Creation.



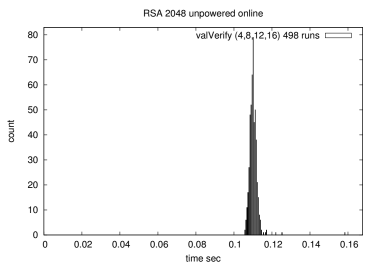
(i) RSA1024, Signature Verification.



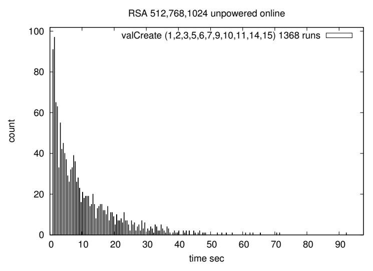
(j) RSA2048, Key Pair Creation.



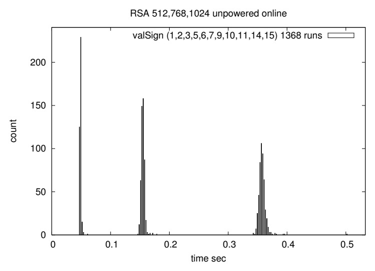
(k) RSA2048, Signature Creation.



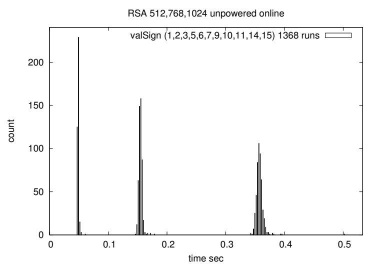
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

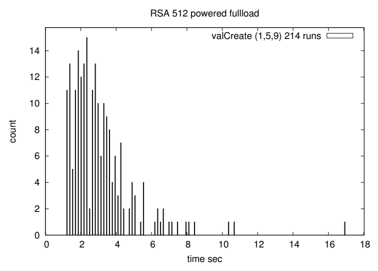


(n) RSA -1024, Signature Creation.

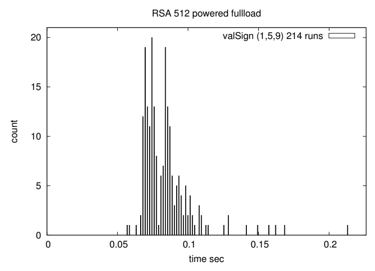


(o) RSA -1024, Signature Verification.

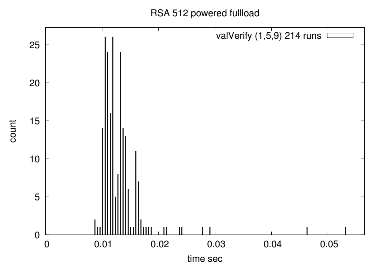
Fig. 39. 5800xm, Online Unpowered, RSA 512, 768, 1024, 2048.



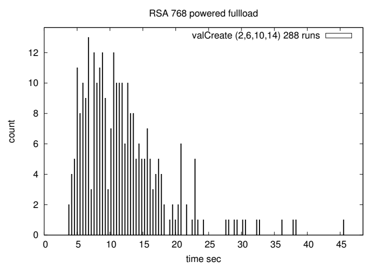
(a) RSA512, Key Pair Creation.



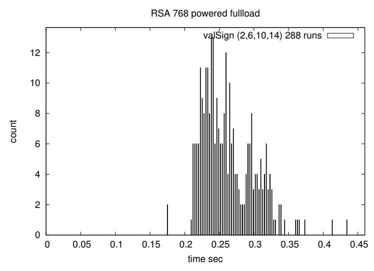
(b) RSA512, Signature Creation.



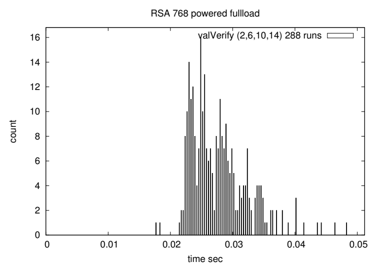
(c) RSA512, Signature Verification.



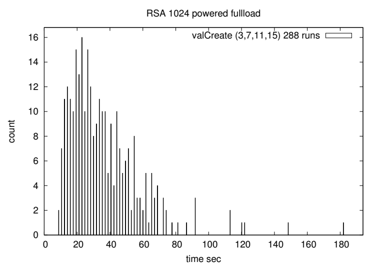
(d) RSA768, Key Pair Creation.



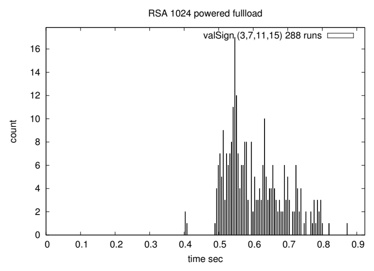
(e) RSA768, Signature Creation.



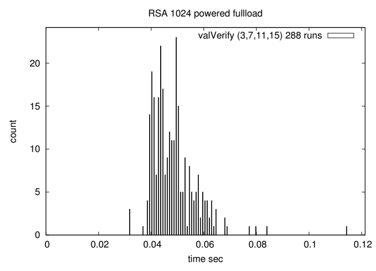
(f) RSA768, Signature Verification.



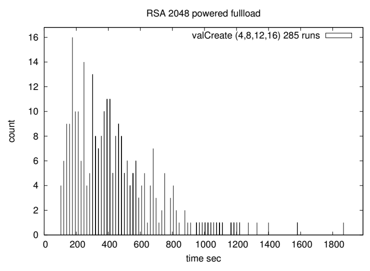
(g) RSA1024, Key Pair Creation.



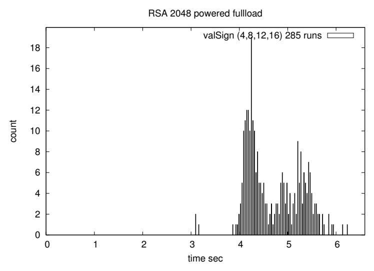
(h) RSA1024, Signature Creation.



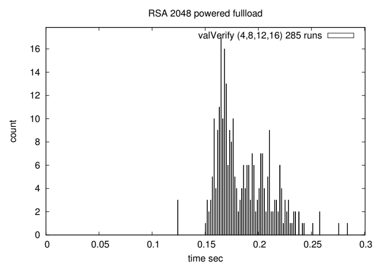
(i) RSA1024, Signature Verification.



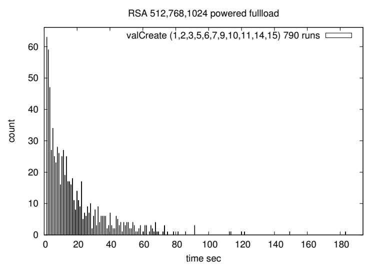
(j) RSA2048, Key Pair Creation.



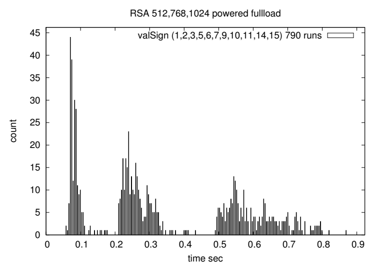
(k) RSA2048, Signature Creation.



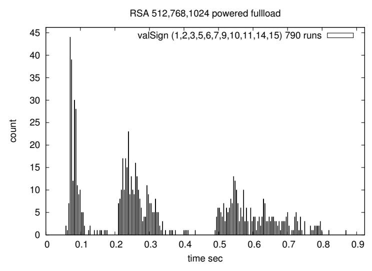
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

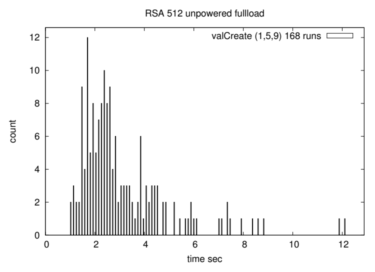


(n) RSA -1024, Signature Creation.

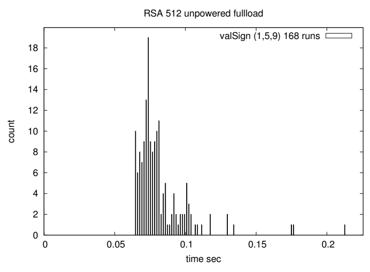


(o) RSA -1024, Signature Verification.

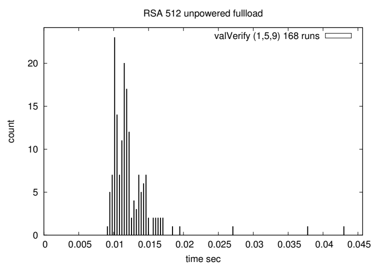
Fig. 40. 5800xm, Full-load Powered, RSA 512, 768, 1024, 2048.



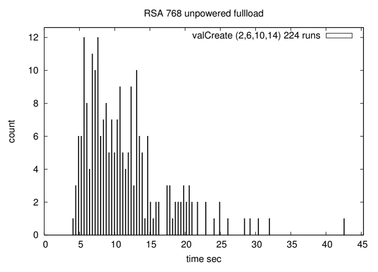
(a) RSA512, Key Pair Creation.



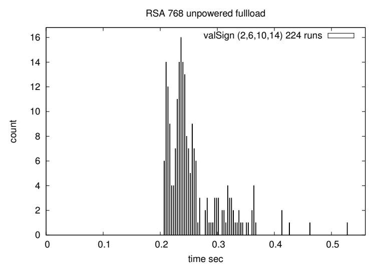
(b) RSA512, Signature Creation.



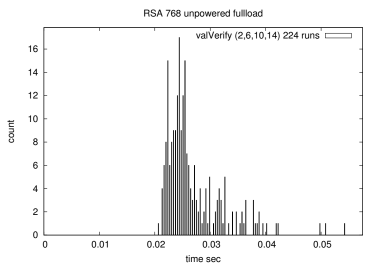
(c) RSA512, Signature Verification.



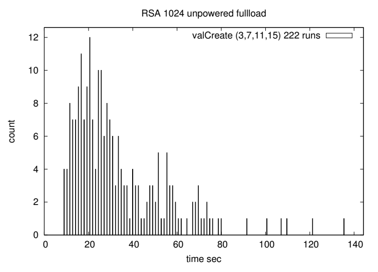
(d) RSA768, Key Pair Creation.



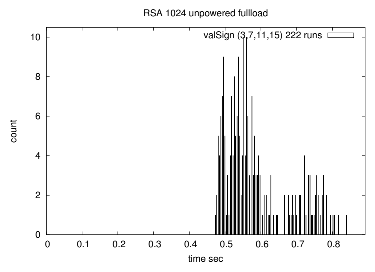
(e) RSA768, Signature Creation.



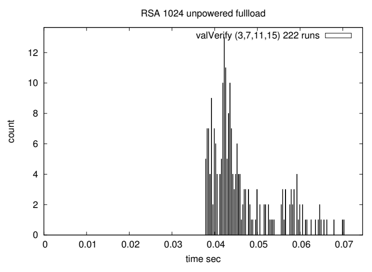
(f) RSA768, Signature Verification.



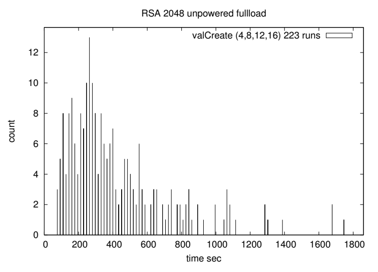
(g) RSA1024, Key Pair Creation.



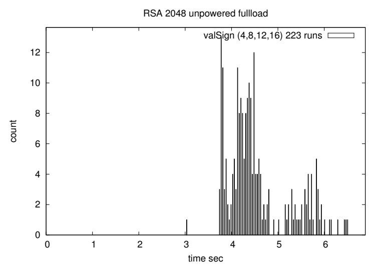
(h) RSA1024, Signature Creation.



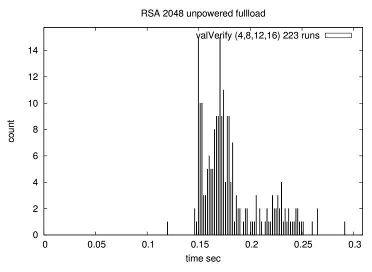
(i) RSA1024, Signature Verification.



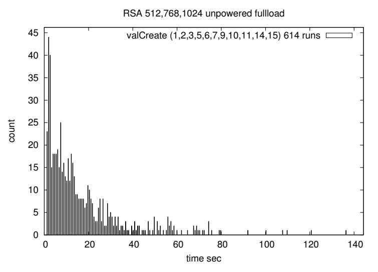
(j) RSA2048, Key Pair Creation.



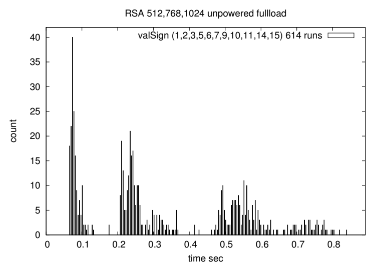
(k) RSA2048, Signature Creation.



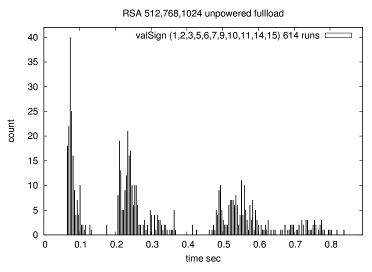
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

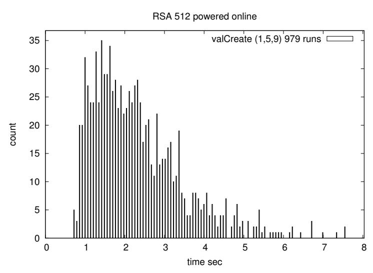


(n) RSA -1024, Signature Creation.

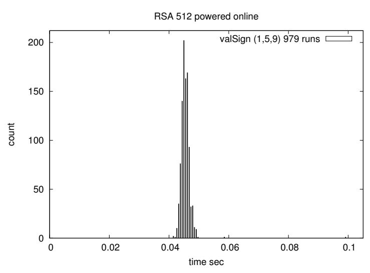


(o) RSA -1024, Signature Verification.

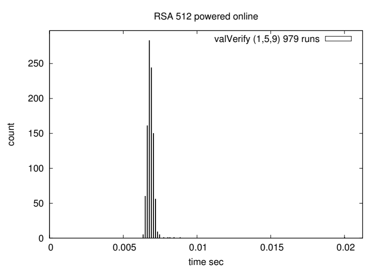
Fig. 41. 5800xm, Full-load Unpowered, RSA 512, 768, 1024, 2048



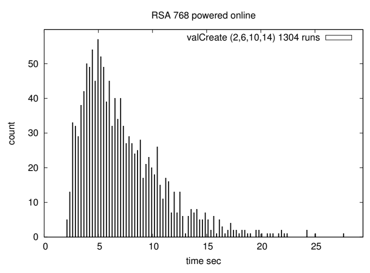
(a) RSA512, Key Pair Creation.



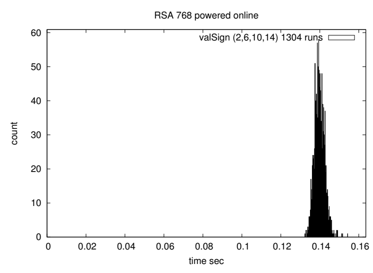
(b) RSA512, Signature Creation.



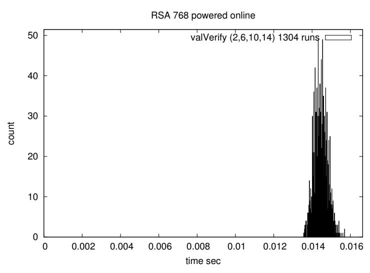
(c) RSA512, Signature Verification.



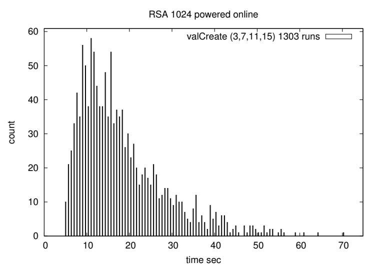
(d) RSA768, Key Pair Creation.



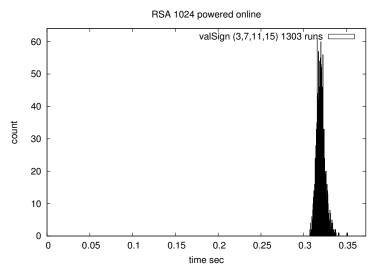
(e) RSA768, Signature Creation.



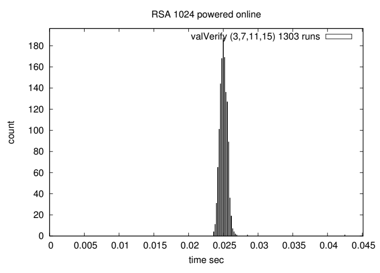
(f) RSA768, Signature Verification.



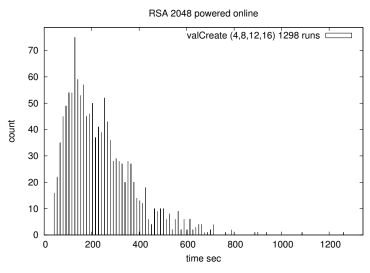
(g) RSA1024, Key Pair Creation.



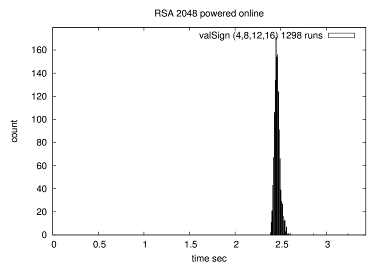
(h) RSA1024, Signature Creation.



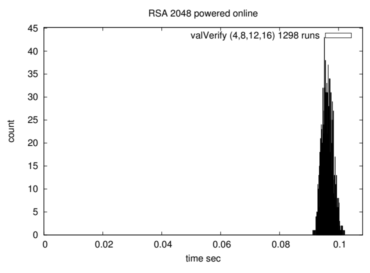
(i) RSA1024, Signature Verification.



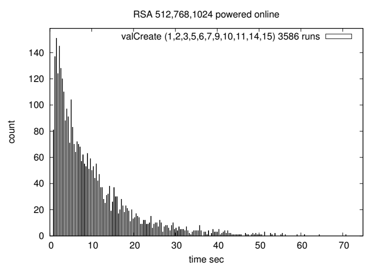
(j) RSA2048, Key Pair Creation.



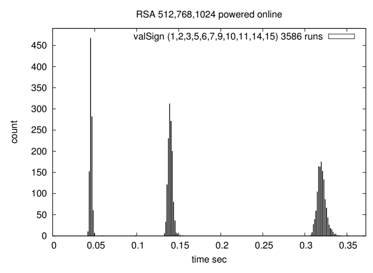
(k) RSA2048, Signature Creation.



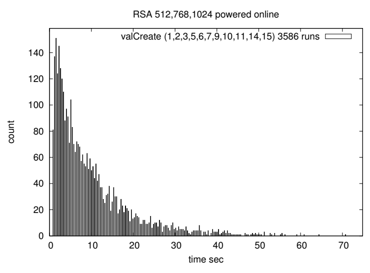
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

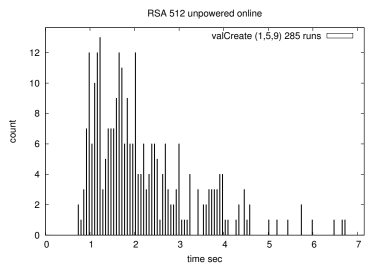


(n) RSA -1024, Signature Creation.

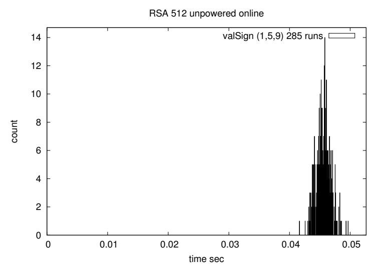


(o) RSA -1024, Signature Verification.

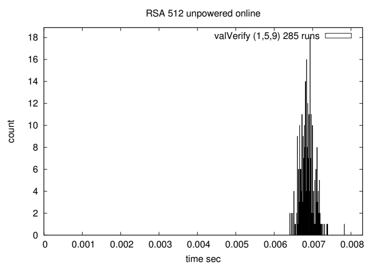
Fig. 42. E50, Online Powered, RSA 512, 768, 1024, 2048.



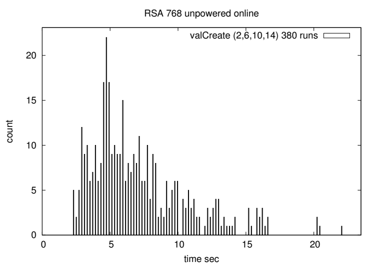
(a) RSA512, Key Pair Creation.



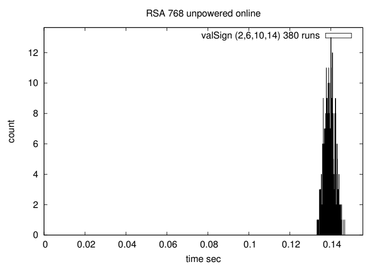
(b) RSA512, Signature Creation.



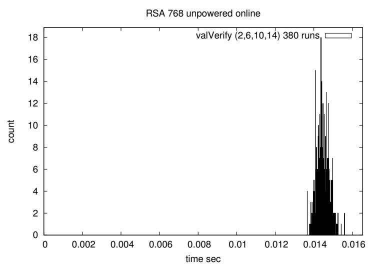
(c) RSA512, Signature Verification.



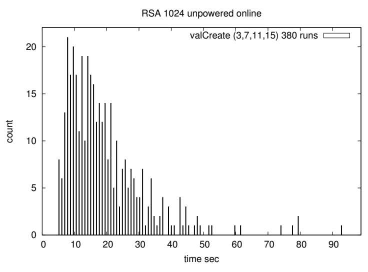
(d) RSA768, Key Pair Creation.



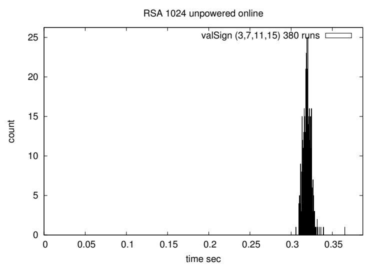
(e) RSA768, Signature Creation.



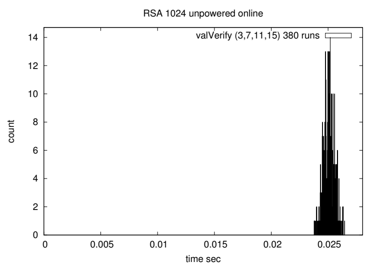
(f) RSA768, Signature Verification.



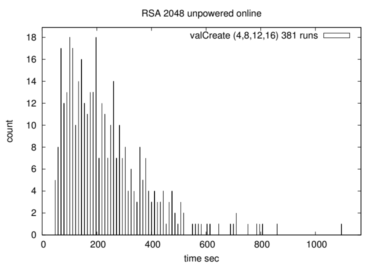
(g) RSA1024, Key Pair Creation.



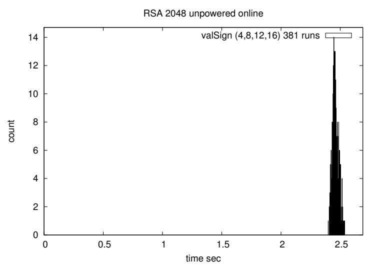
(h) RSA1024, Signature Creation.



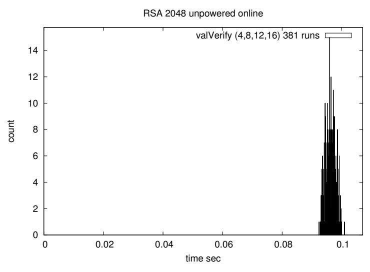
(i) RSA1024, Signature Verification.



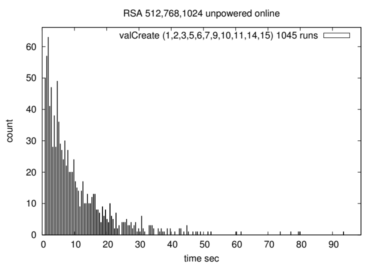
(j) RSA2048, Key Pair Creation.



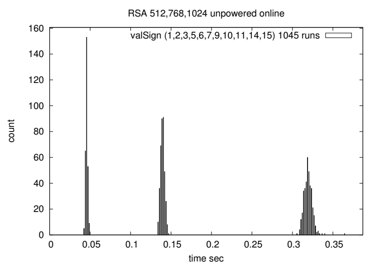
(k) RSA2048, Signature Creation.



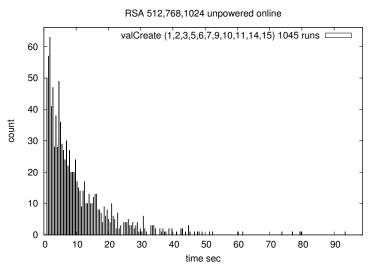
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

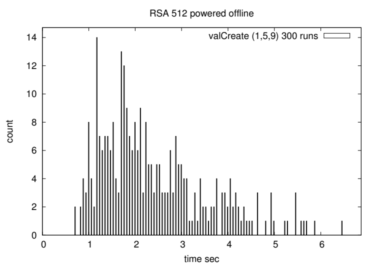


(n) RSA -1024, Signature Creation.

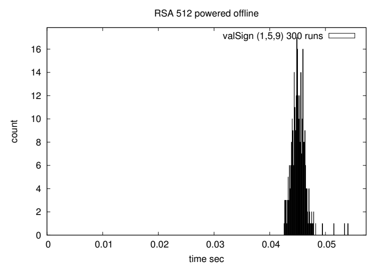


(o) RSA -1024, Signature Verification.

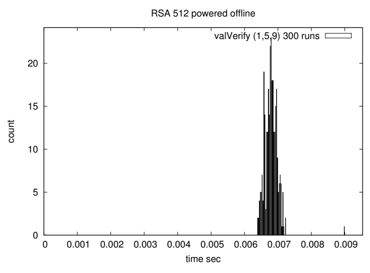
Fig. 43. E50, Online Unpowered, RSA 512, 768, 1024, 2048.



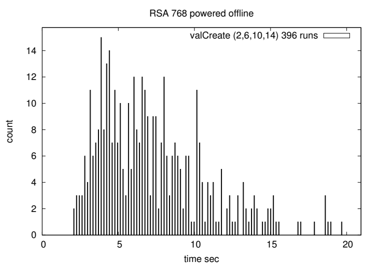
(a) RSA512, Key Pair Creation.



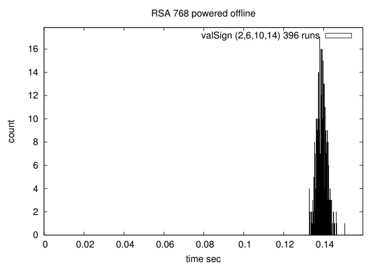
(b) RSA512, Signature Creation.



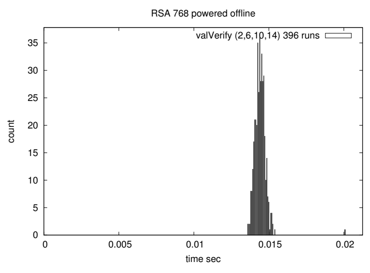
(c) RSA512, Signature Verification.



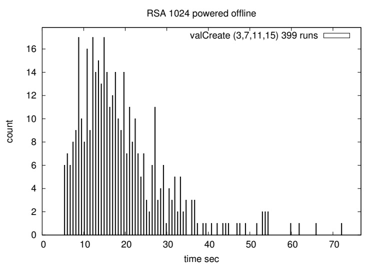
(d) RSA768, Key Pair Creation.



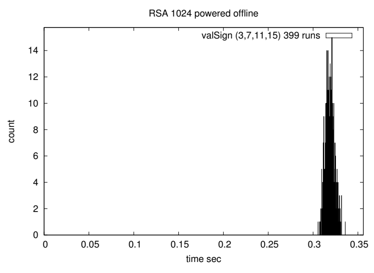
(e) RSA768, Signature Creation.



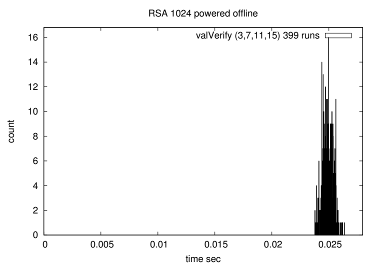
(f) RSA768, Signature Verification.



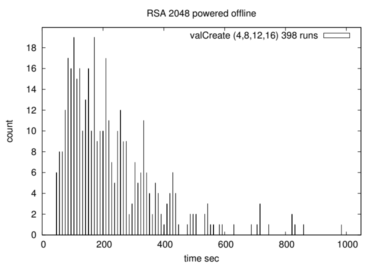
(g) RSA1024, Key Pair Creation.



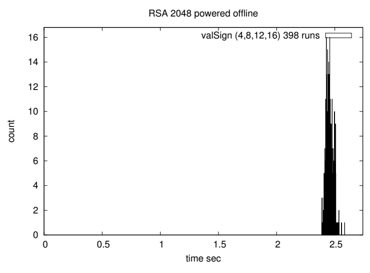
(h) RSA1024, Signature Creation.



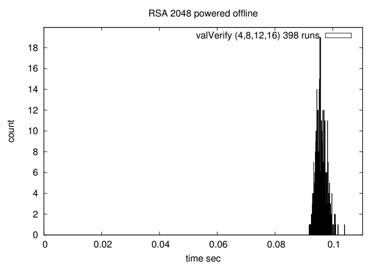
(i) RSA1024, Signature Verification.



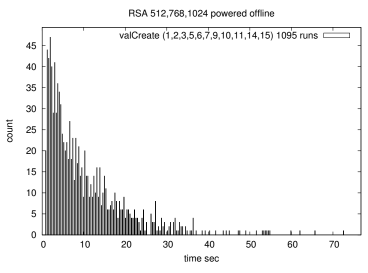
(j) RSA2048, Key Pair Creation.



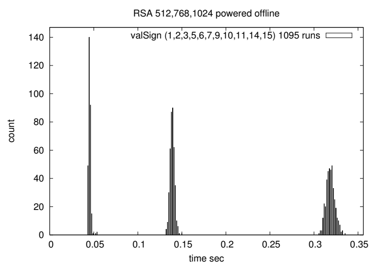
(k) RSA2048, Signature Creation.



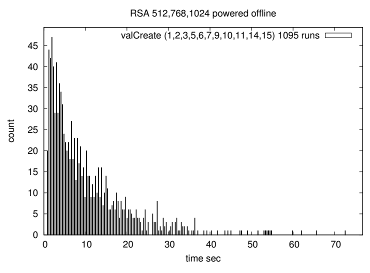
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

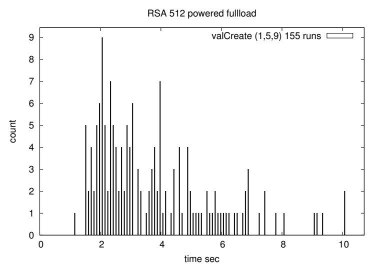


(n) RSA -1024, Signature Creation.

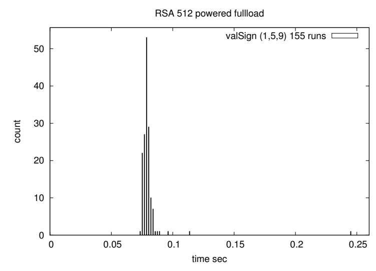


(o) RSA -1024, Signature Verification.

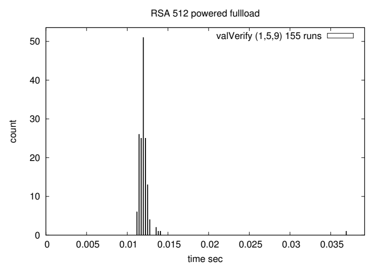
Fig. 44. E50, Offline Powered, RSA 512, 768, 1024, 2048.



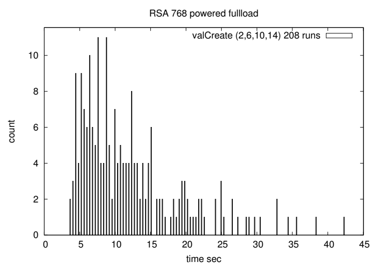
(a) RSA512, Key Pair Creation.



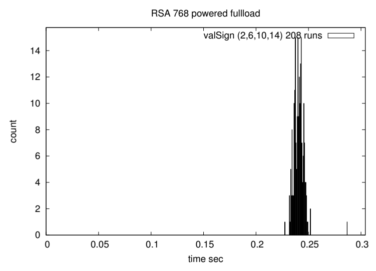
(b) RSA512, Signature Creation.



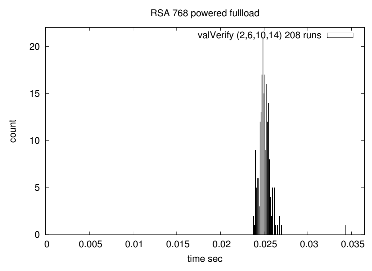
(c) RSA512, Signature Verification.



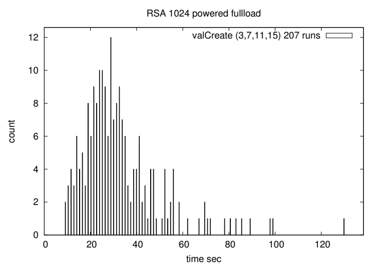
(d) RSA768, Key Pair Creation.



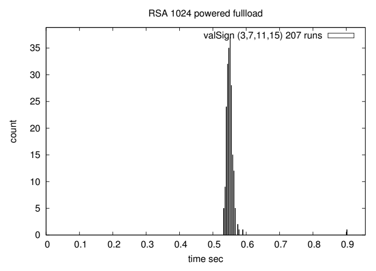
(e) RSA768, Signature Creation.



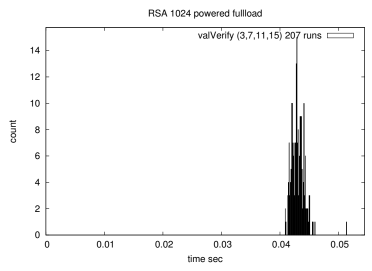
(f) RSA768, Signature Verification.



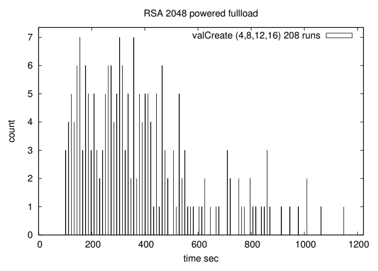
(g) RSA1024, Key Pair Creation.



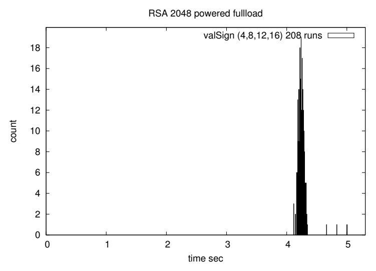
(h) RSA1024, Signature Creation.



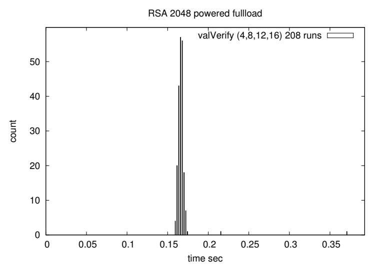
(i) RSA1024, Signature Verification.



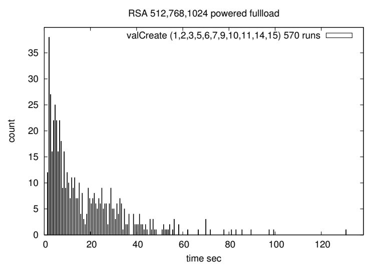
(j) RSA2048, Key Pair Creation.



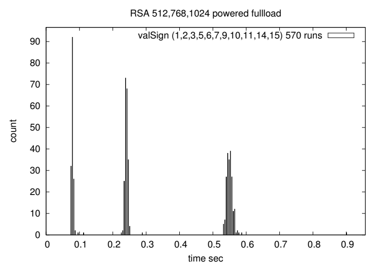
(k) RSA2048, Signature Creation.



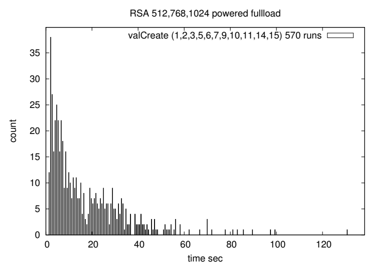
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

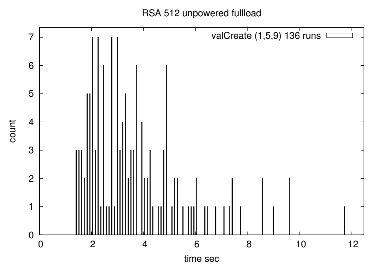


(n) RSA -1024, Signature Creation.

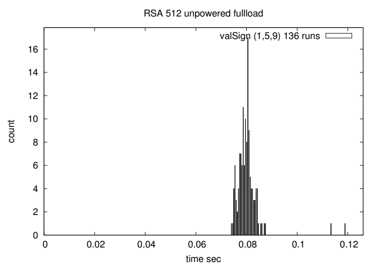


(o) RSA -1024, Signature Verification.

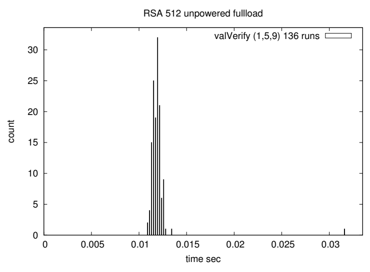
Fig. 45. E50, Full-load Powered, RSA 512, 768, 1024, 2048.



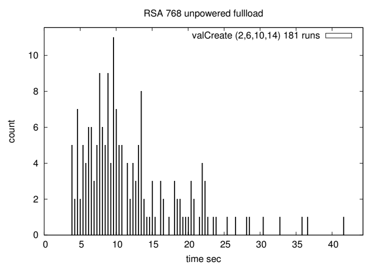
(a) RSA512, Key Pair Creation.



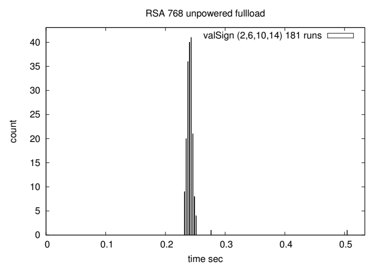
(b) RSA512, Signature Creation.



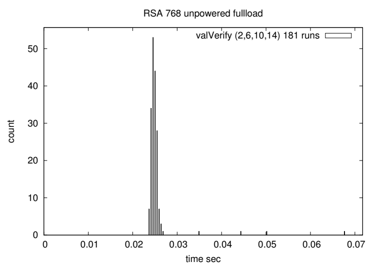
(c) RSA512, Signature Verification.



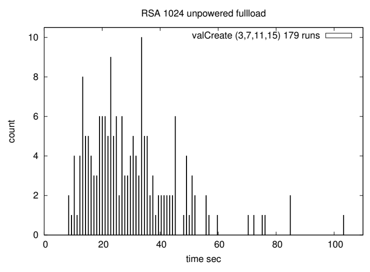
(d) RSA768, Key Pair Creation.



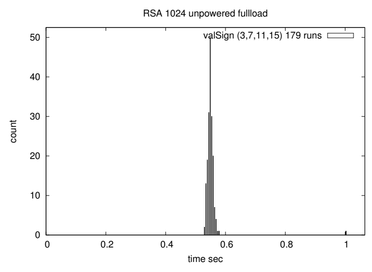
(e) RSA768, Signature Creation.



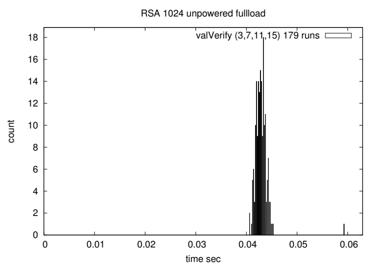
(f) RSA768, Signature Verification.



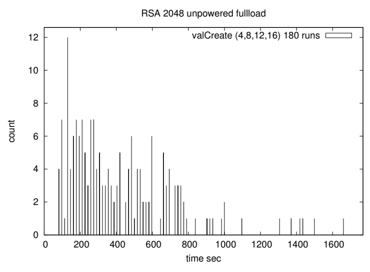
(g) RSA1024, Key Pair Creation.



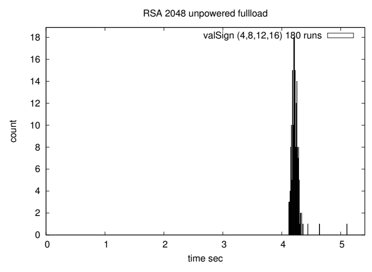
(h) RSA1024, Signature Creation.



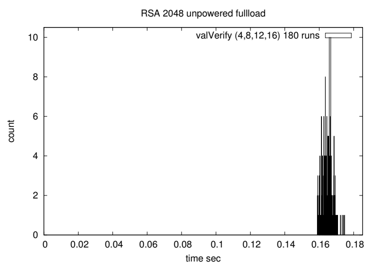
(i) RSA1024, Signature Verification.



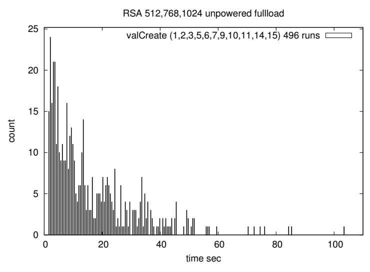
(j) RSA2048, Key Pair Creation.



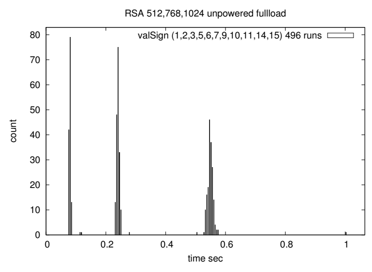
(k) RSA2048, Signature Creation.



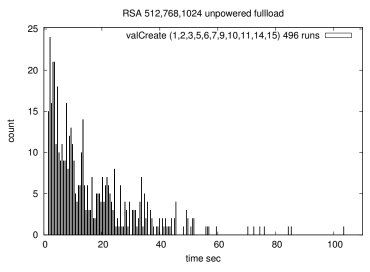
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

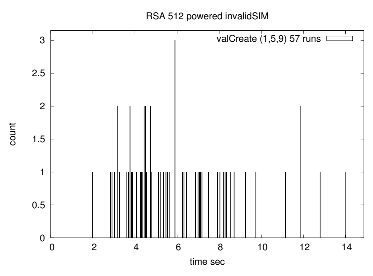


(n) RSA -1024, Signature Creation.

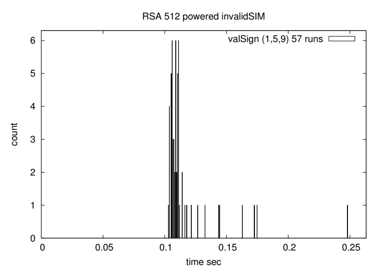


(o) RSA -1024, Signature Verification.

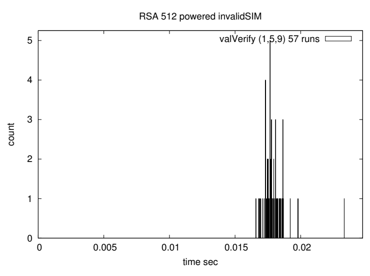
Fig. 46. E50, Full-load Unpowered, RSA 512, 768, 1024, 2048.



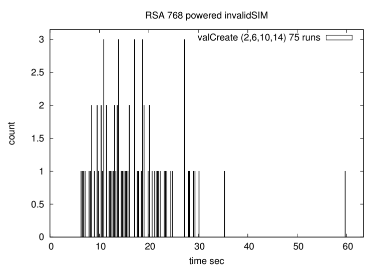
(a) RSA512, Key Pair Creation.



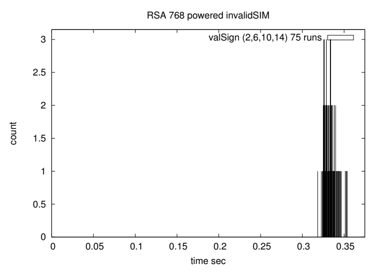
(b) RSA512, Signature Creation.



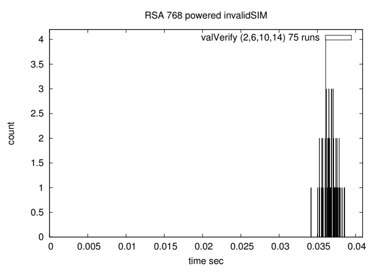
(c) RSA512, Signature Verification.



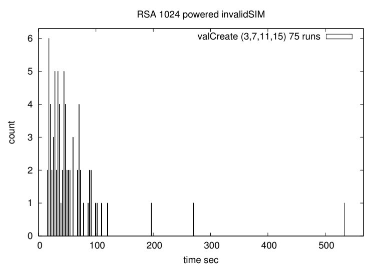
(d) RSA768, Key Pair Creation.



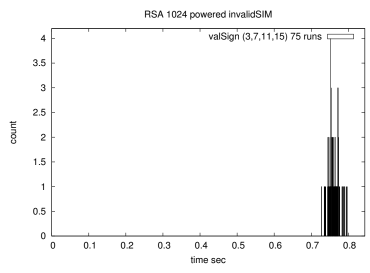
(e) RSA768, Signature Creation.



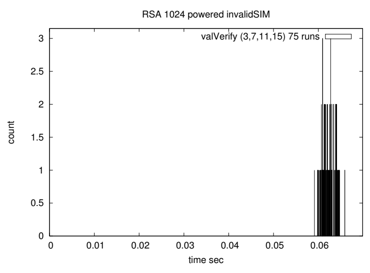
(f) RSA768, Signature Verification.



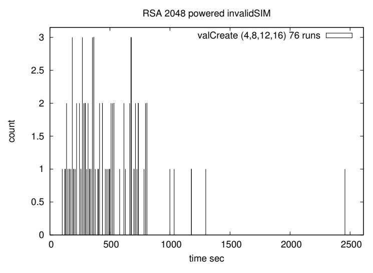
(g) RSA1024, Key Pair Creation.



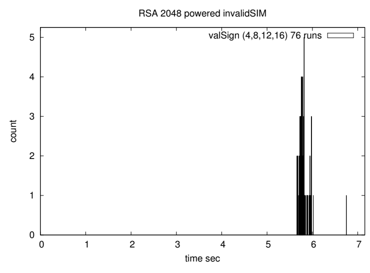
(h) RSA1024, Signature Creation.



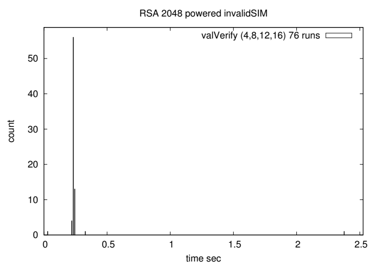
(i) RSA1024, Signature Verification.



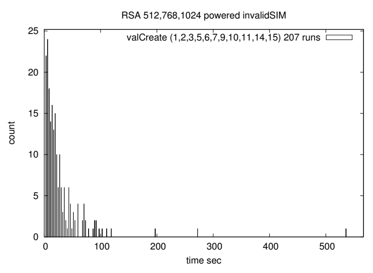
(j) RSA2048, Key Pair Creation.



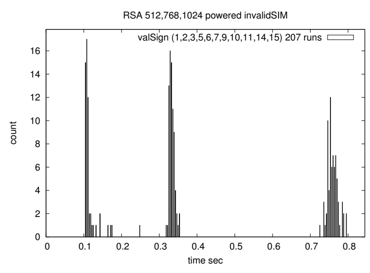
(k) RSA2048, Signature Creation.



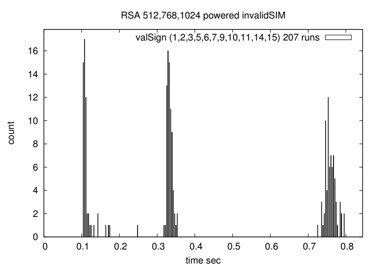
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

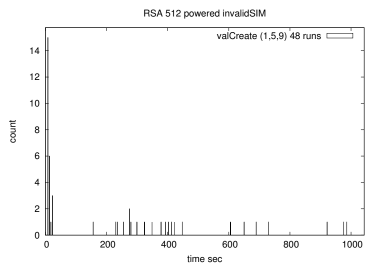


(n) RSA -1024, Signature Creation.

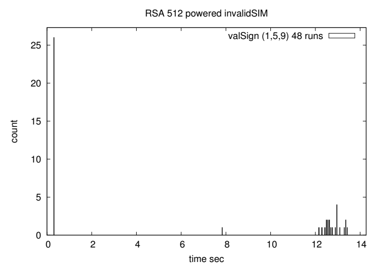


(o) RSA -1024, Signature Verification.

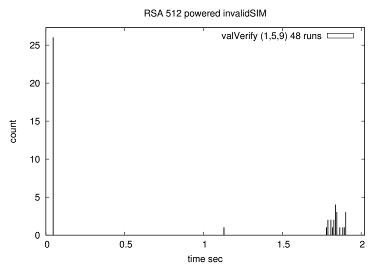
Fig. 47. S65, InvalidSIM Powered, RSA 512, 768, 1024, 2048.



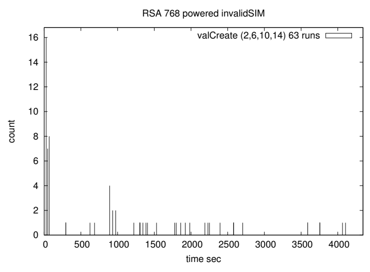
(a) RSA512, Key Pair Creation.



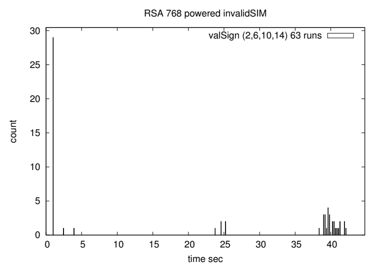
(b) RSA512, Signature Creation.



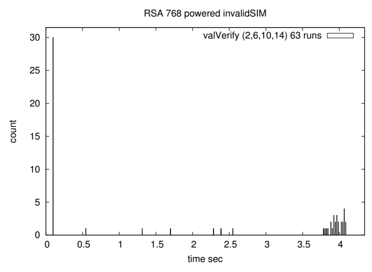
(c) RSA512, Signature Verification.



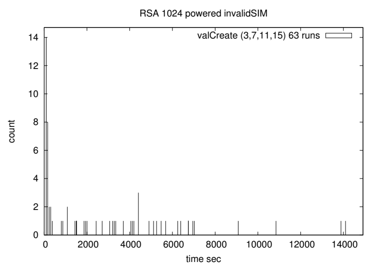
(d) RSA768, Key Pair Creation.



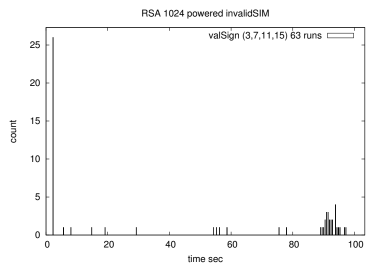
(e) RSA768, Signature Creation.



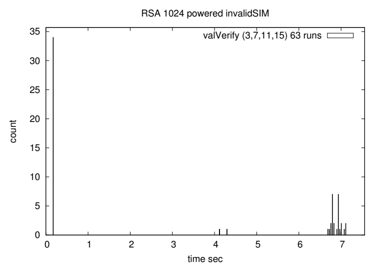
(f) RSA768, Signature Verification.



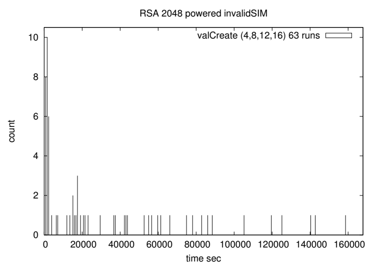
(g) RSA1024, Key Pair Creation.



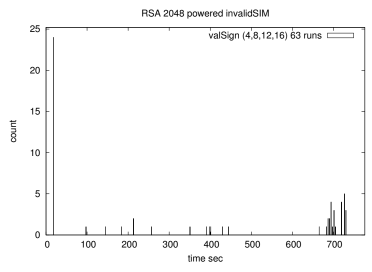
(h) RSA1024, Signature Creation.



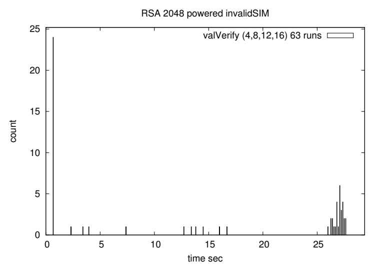
(i) RSA1024, Signature Verification.



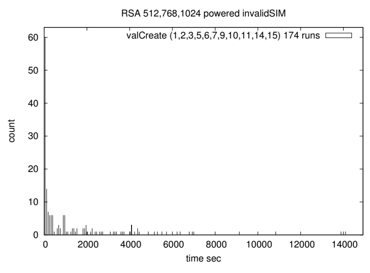
(j) RSA2048, Key Pair Creation.



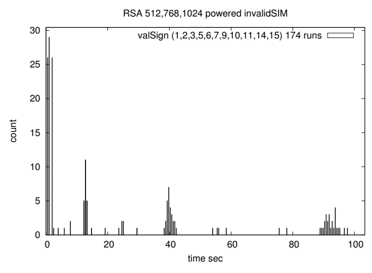
(k) RSA2048, Signature Creation.



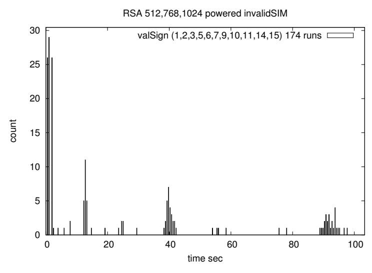
(l) RSA2048, Signature Verification.



(m) RSA -1024, Key Pair Creation.

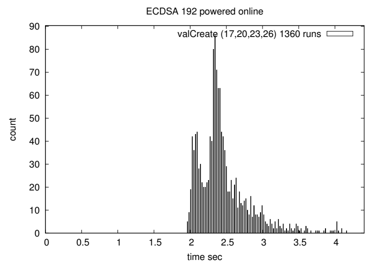


(n) RSA -1024, Signature Creation.

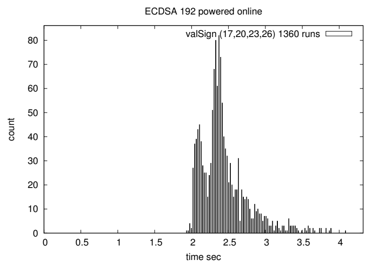


(o) RSA -1024, Signature Verification.

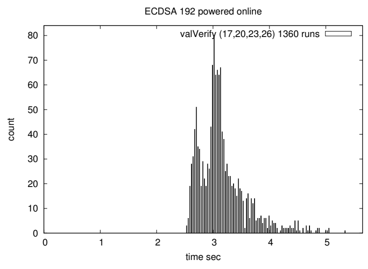
Fig. 48. K750i, InvalidSIM Powered, RSA 512, 768, 1024, 2048.



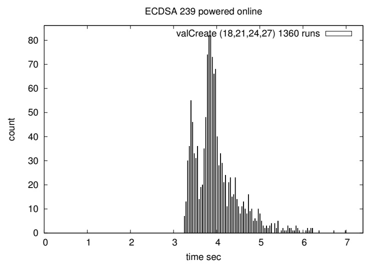
(a) ECDSA192, Key Pair Creation.



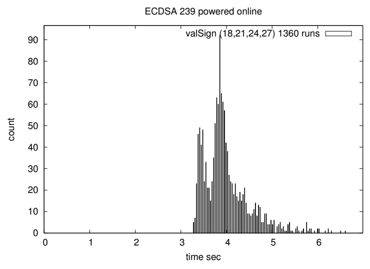
(b) ECDSA192, Signature Creation.



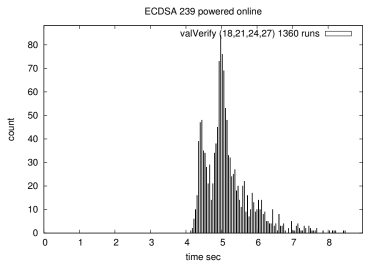
(c) ECDSA192, Signature Verification.



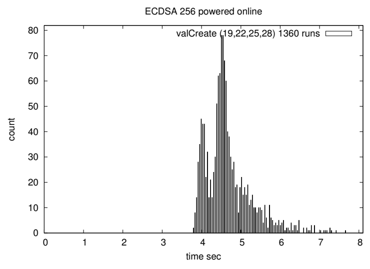
(d) ECDSA239, Key Pair Creation.



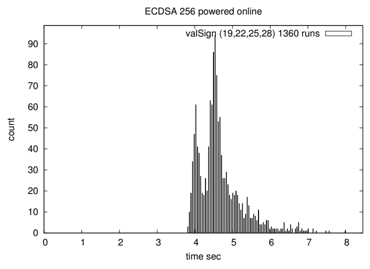
(e) ECDSA239, Signature Creation.



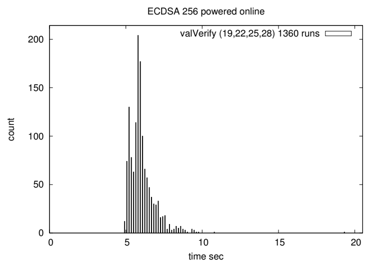
(f) ECDSA239, Signature Verification.



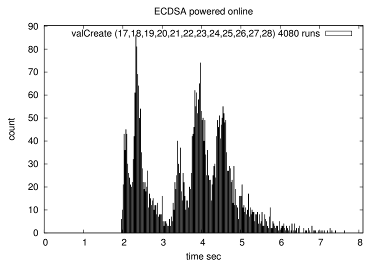
(g) ECDSA256, Key Pair Creation.



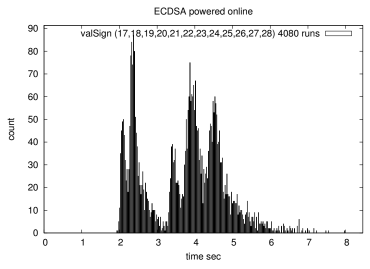
(h) ECDSA256, Signature Creation.



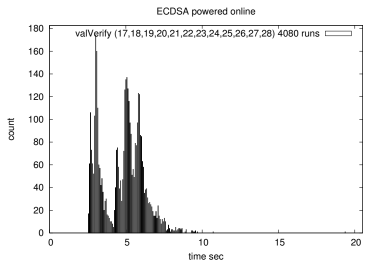
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

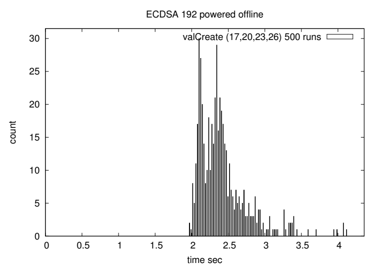


(k) ECDSA, Signature Creation.

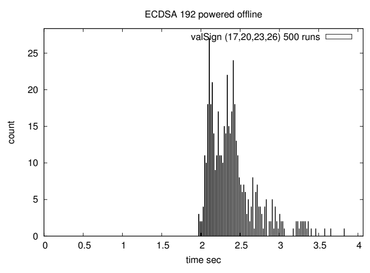


(l) ECDSA, Signature Verification.

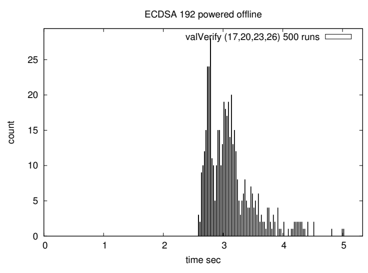
Fig. 49. M600i, Online Powered, ECDSA 192, 239 and 256.



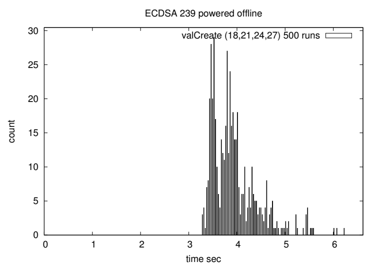
(a) ECDSA192, Key Pair Creation.



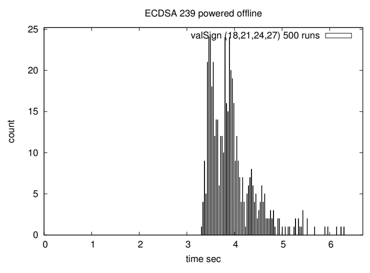
(b) ECDSA192, Signature Creation.



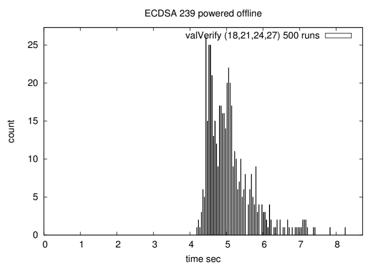
(c) ECDSA192, Signature Verification.



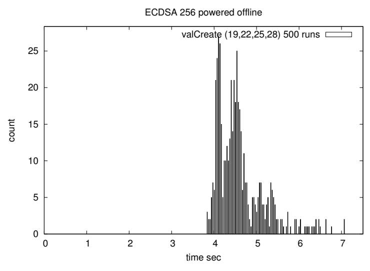
(d) ECDSA239, Key Pair Creation.



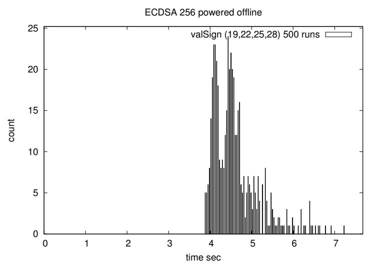
(e) ECDSA239, Signature Creation.



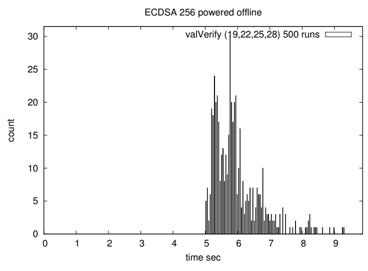
(f) ECDSA239, Signature Verification.



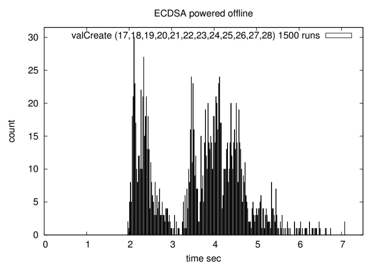
(g) ECDSA256, Key Pair Creation.



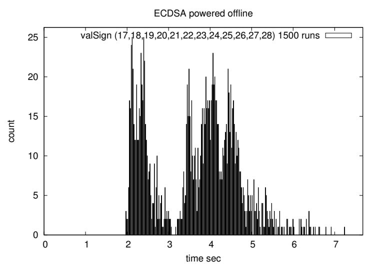
(h) ECDSA256, Signature Creation.



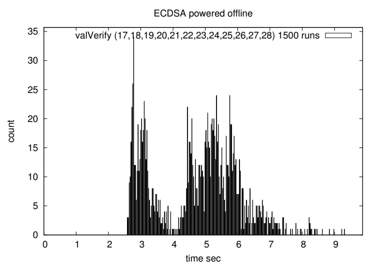
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

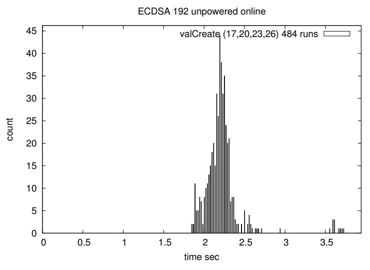


(k) ECDSA, Signature Creation.

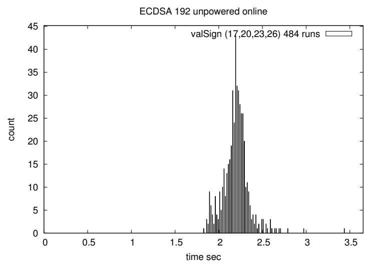


(l) ECDSA, Signature Verification.

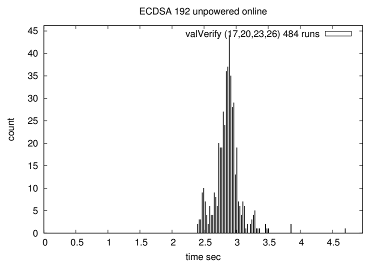
Fig. 50. M600i, Offline Powered, ECDSA 192, 239, 256.



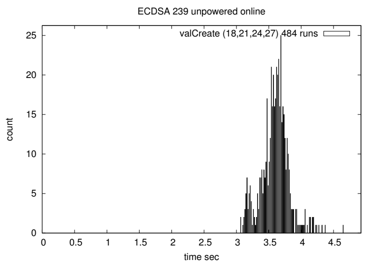
(a) ECDSA192, Key Pair Creation.



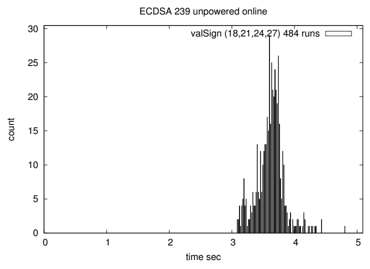
(b) ECDSA192, Signature Creation.



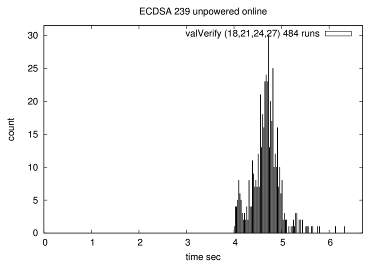
(c) ECDSA192, Signature Verification.



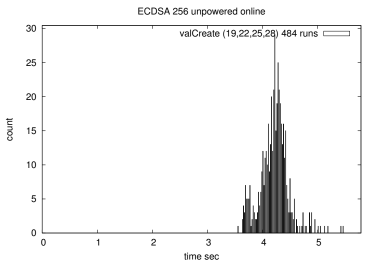
(d) ECDSA239, Key Pair Creation.



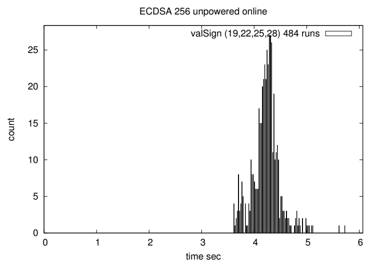
(e) ECDSA239, Signature Creation.



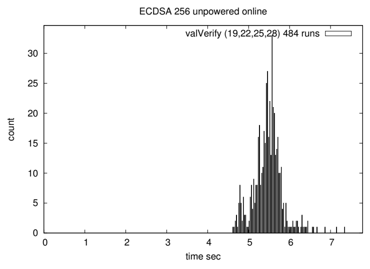
(f) ECDSA239, Signature Verification.



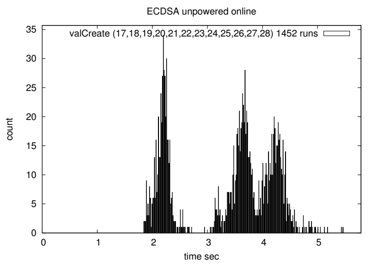
(g) ECDSA256, Key Pair Creation.



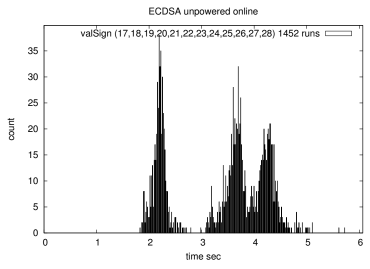
(h) ECDSA256, Signature Creation.



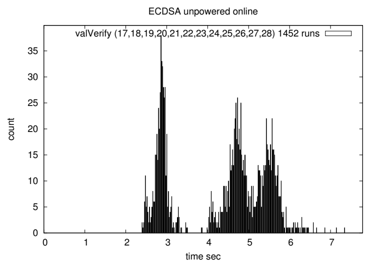
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

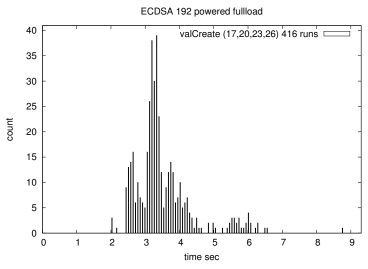


(k) ECDSA, Signature Creation.

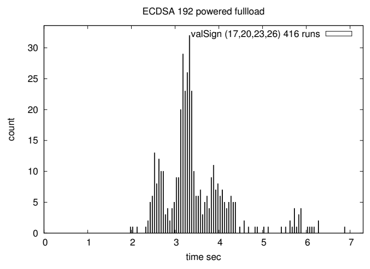


(l) ECDSA, Signature Verification.

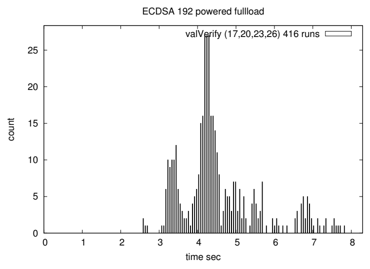
Fig. 51. M600i, Online Unpowered, ECDSA 192, 239, 256.



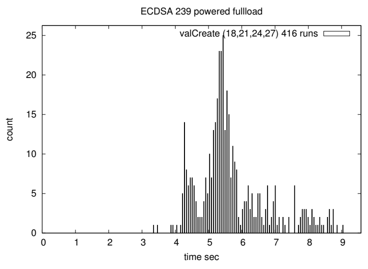
(a) ECDSA192, Key Pair Creation.



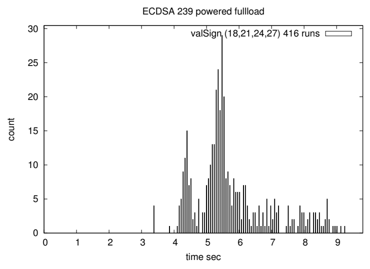
(b) ECDSA192, Signature Creation.



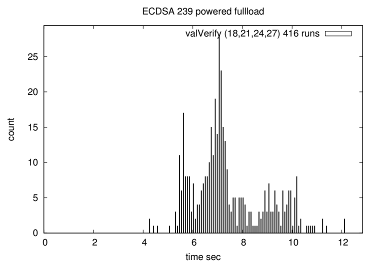
(c) ECDSA192, Signature Verification.



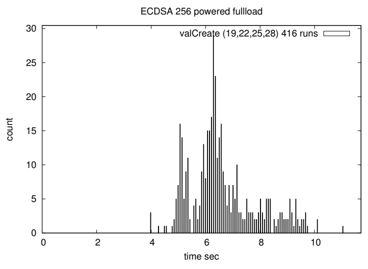
(d) ECDSA239, Key Pair Creation.



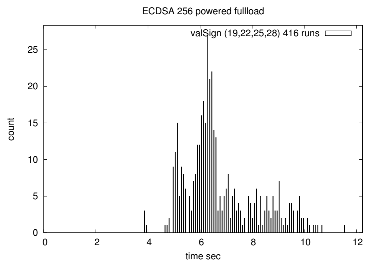
(e) ECDSA239, Signature Creation.



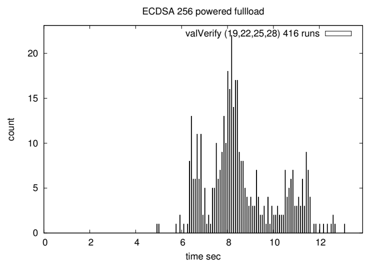
(f) ECDSA239, Signature Verification.



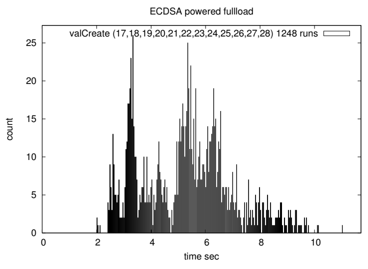
(g) ECDSA256, Key Pair Creation.



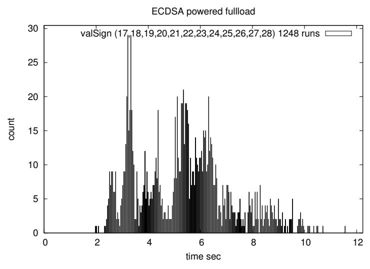
(h) ECDSA256, Signature Creation.



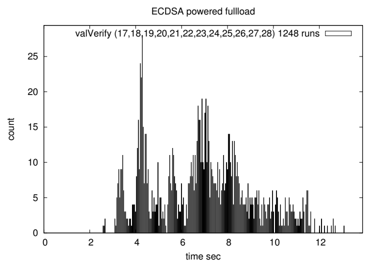
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

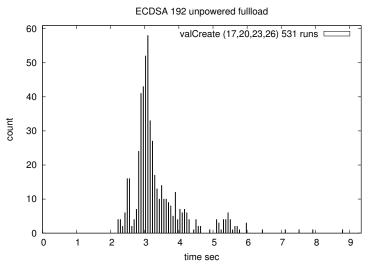


(k) ECDSA, Signature Creation.

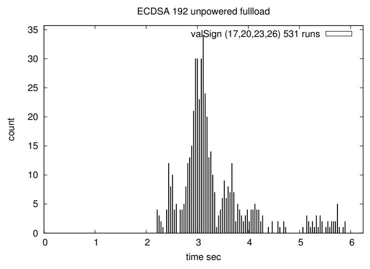


(l) ECDSA, Signature Verification.

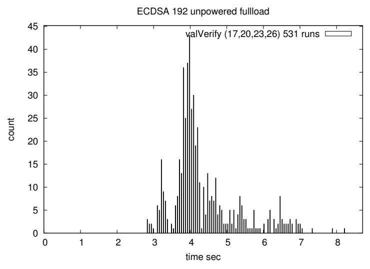
Fig. 52. M600i, Full-load Powered, ECDSA 192,239,256.



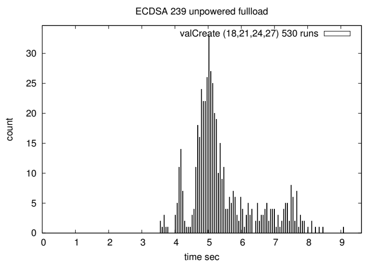
(a) ECDSA192, Key Pair Creation.



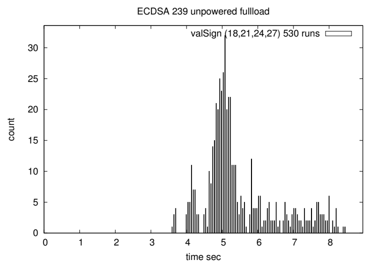
(b) ECDSA192, Signature Creation.



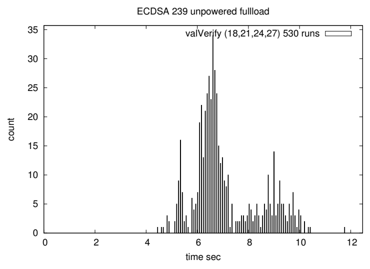
(c) ECDSA192, Signature Verification.



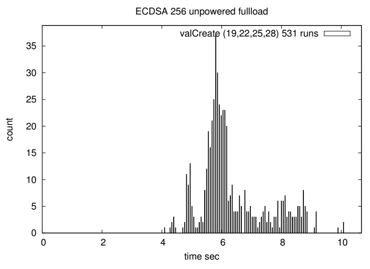
(d) ECDSA239, Key Pair Creation.



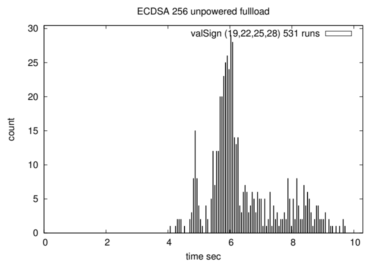
(e) ECDSA239, Signature Creation.



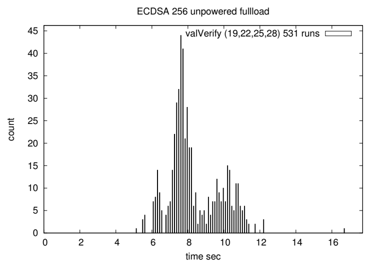
(f) ECDSA239, Signature Verification.



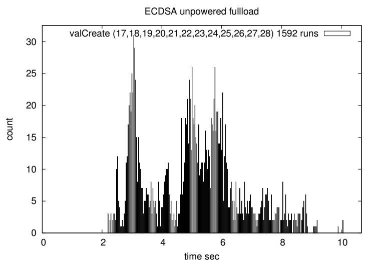
(g) ECDSA256, Key Pair Creation.



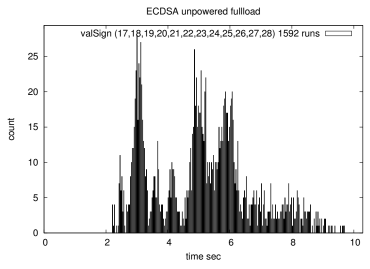
(h) ECDSA256, Signature Creation.



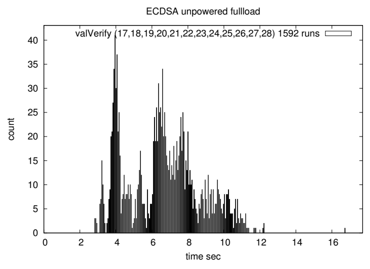
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

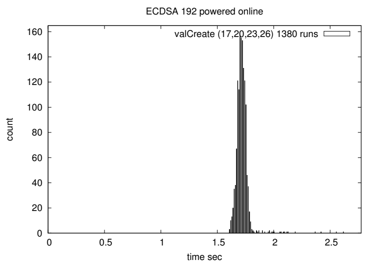


(k) ECDSA, Signature Creation.

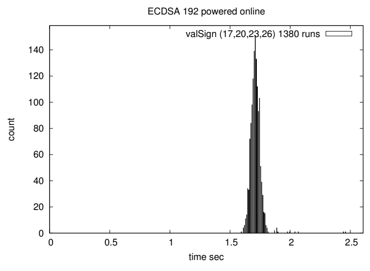


(l) ECDSA, Signature Verification.

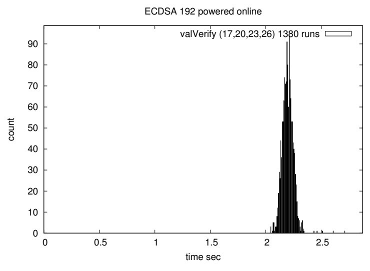
Fig. 53. M600i, Full-load Unpowered, ECDSA 192, 239, 256.



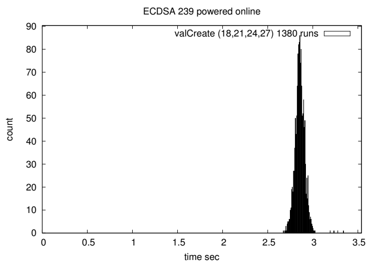
(a) ECDSA192, Key Pair Creation.



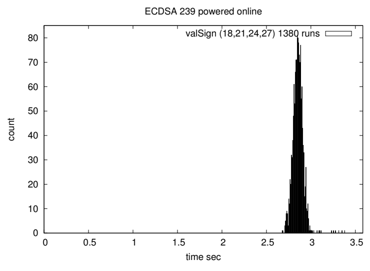
(b) ECDSA192, Signature Creation.



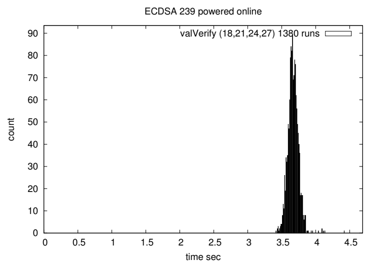
(c) ECDSA192, Signature Verification.



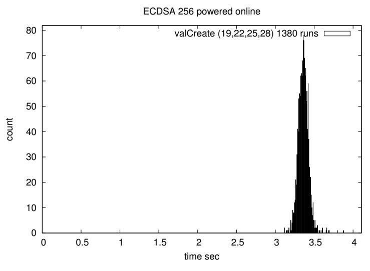
(d) ECDSA239, Key Pair Creation.



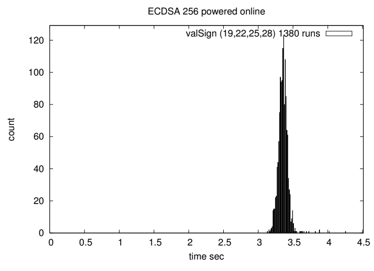
(e) ECDSA239, Signature Creation.



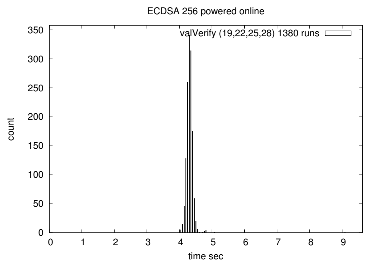
(f) ECDSA239, Signature Verification.



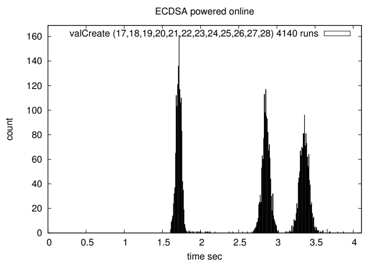
(g) ECDSA256, Key Pair Creation.



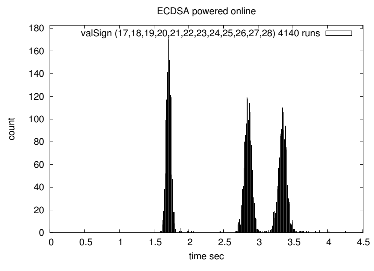
(h) ECDSA256, Signature Creation.



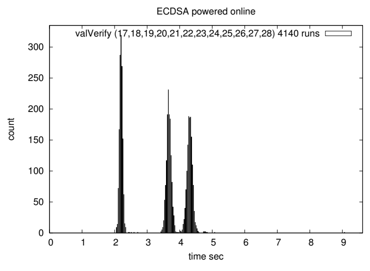
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

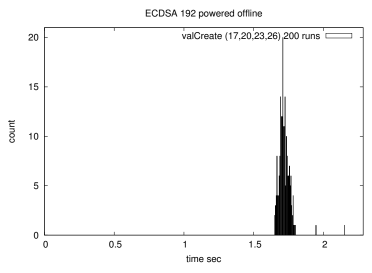


(k) ECDSA, Signature Creation.

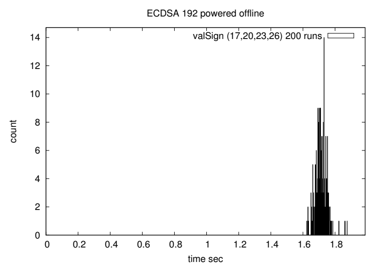


(l) ECDSA, Signature Verification.

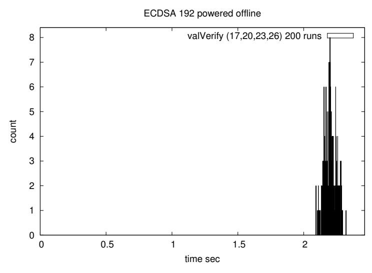
Fig. 54. 5800xm, Online Powered, ECDSA 192, 239, 256.



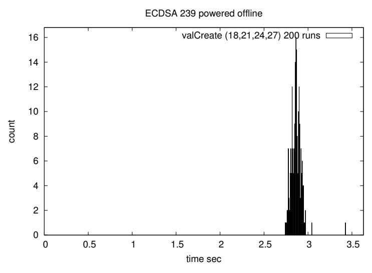
(a) ECDSA192, Key Pair Creation.



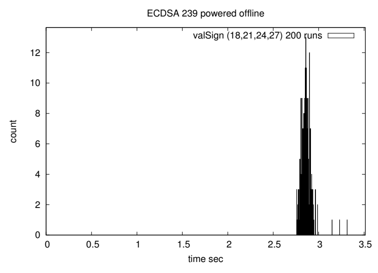
(b) ECDSA192, Signature Creation.



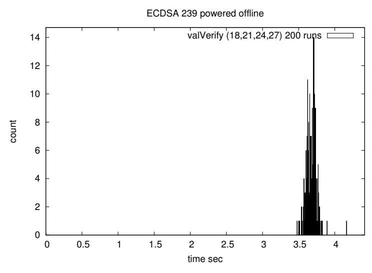
(c) ECDSA192, Signature Verification.



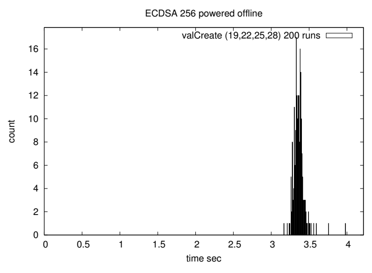
(d) ECDSA239, Key Pair Creation.



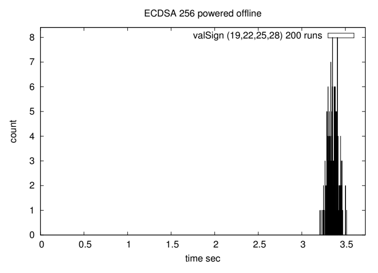
(e) ECDSA239, Signature Creation.



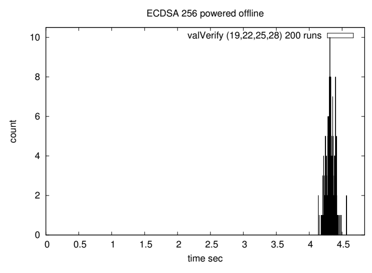
(f) ECDSA239, Signature Verification.



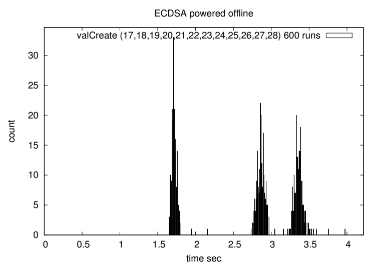
(g) ECDSA256, Key Pair Creation.



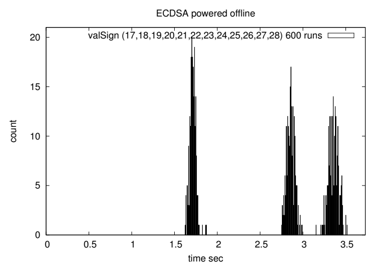
(h) ECDSA256, Signature Creation.



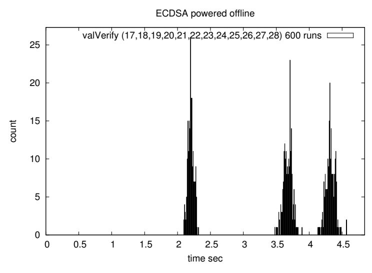
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

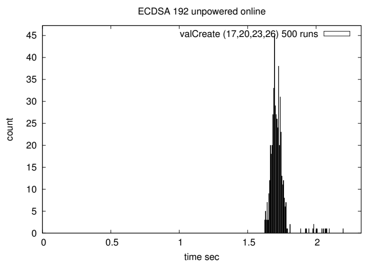


(k) ECDSA, Signature Creation.

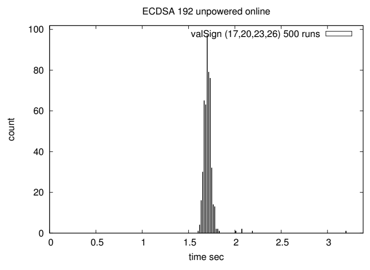


(l) ECDSA, Signature Verification.

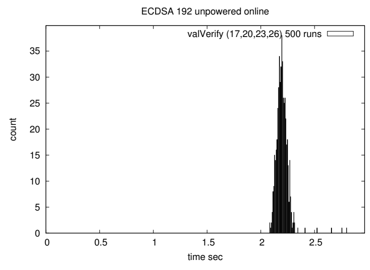
Fig. 55. 5800xm, Offline Powered, ECDSA 192, 239, 256.



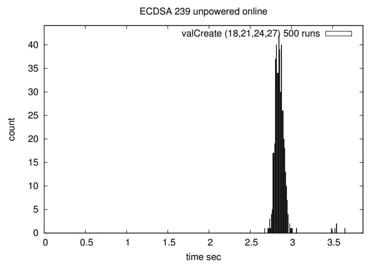
(a) ECDSA192, Key Pair Creation.



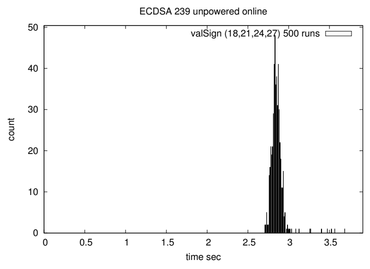
(b) ECDSA192, Signature Creation.



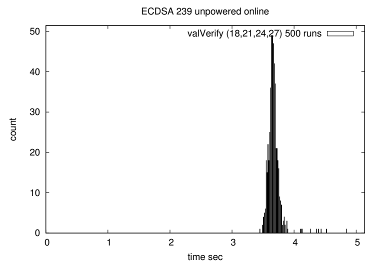
(c) ECDSA192, Signature Verification.



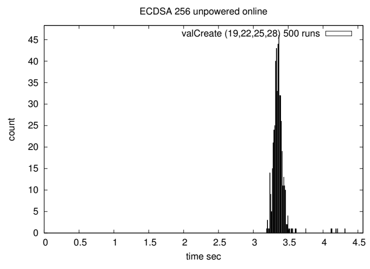
(d) ECDSA239, Key Pair Creation.



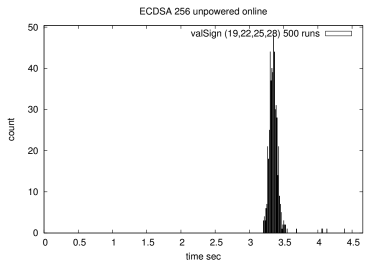
(e) ECDSA239, Signature Creation.



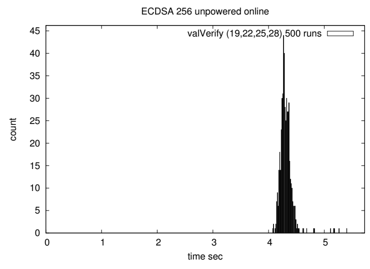
(f) ECDSA239, Signature Verification.



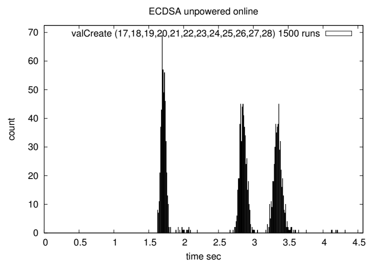
(g) ECDSA256, Key Pair Creation.



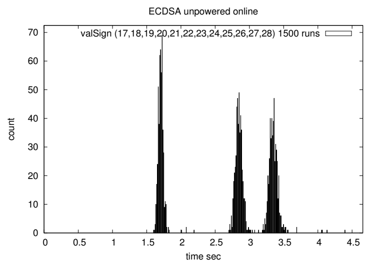
(h) ECDSA256, Signature Creation.



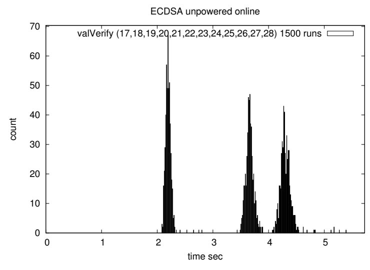
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

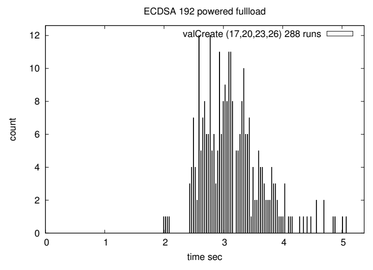


(k) ECDSA, Signature Creation.

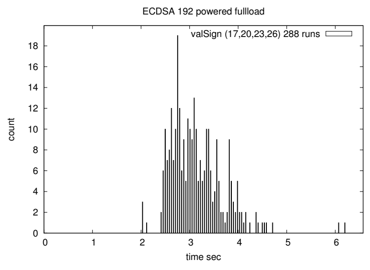


(l) ECDSA, Signature Verification.

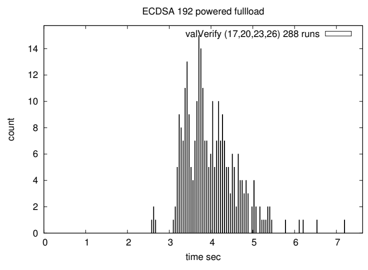
Fig. 56. 5800xm, Online Unpowered, ECDSA 192, 239, 256.



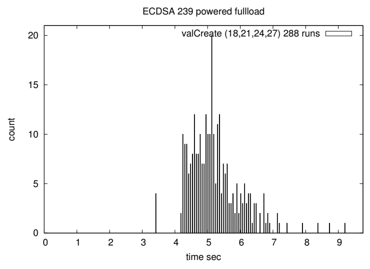
(a) ECDSA192, Key Pair Creation.



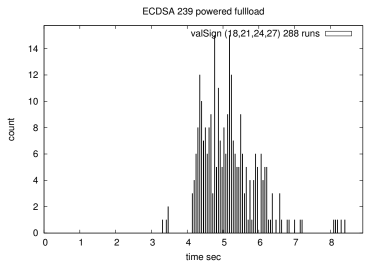
(b) ECDSA192, Signature Creation.



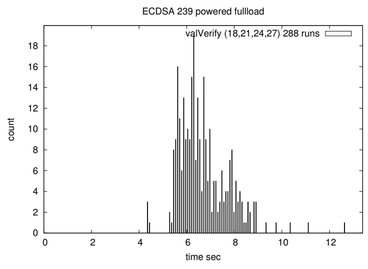
(c) ECDSA192, Signature Verification.



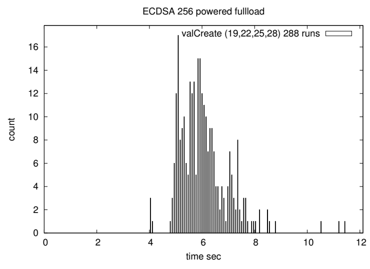
(d) ECDSA239, Key Pair Creation.



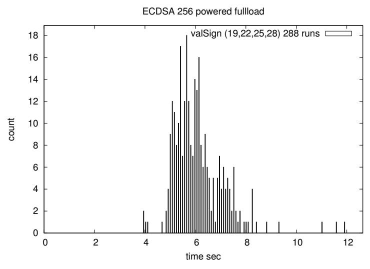
(e) ECDSA239, Signature Creation.



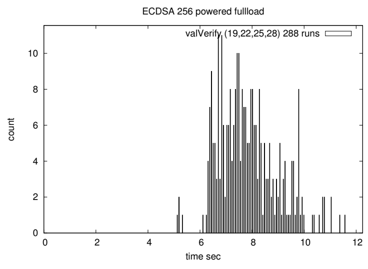
(f) ECDSA239, Signature Verification.



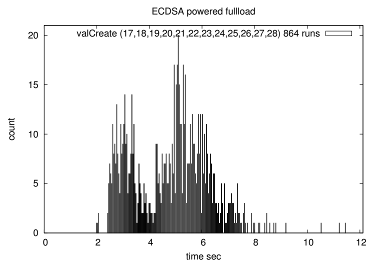
(g) ECDSA256, Key Pair Creation.



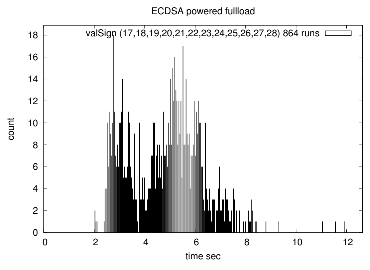
(h) ECDSA256, Signature Creation.



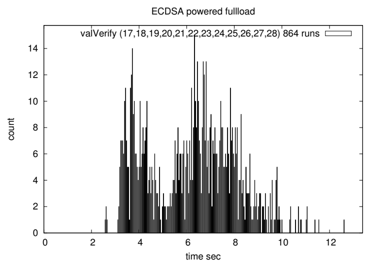
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

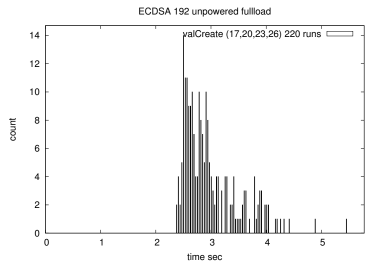


(k) ECDSA, Signature Creation.

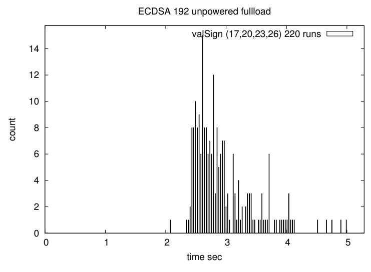


(l) ECDSA, Signature Verification.

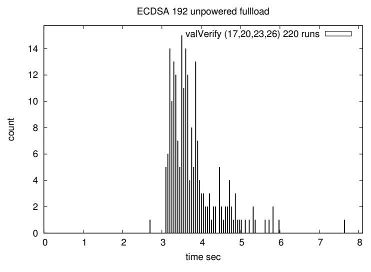
Fig. 57. 5800xm, Fullload Powered, ECDSA 192, 239, 256.



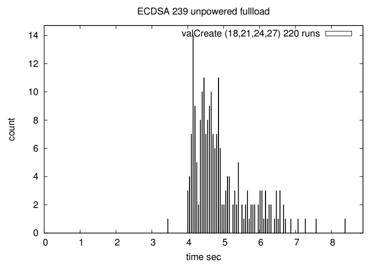
(a) ECDSA192, Key Pair Creation.



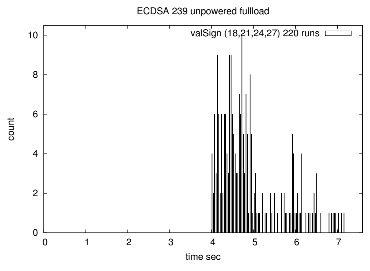
(b) ECDSA192, Signature Creation.



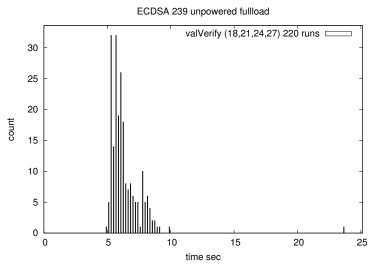
(c) ECDSA192, Signature Verification.



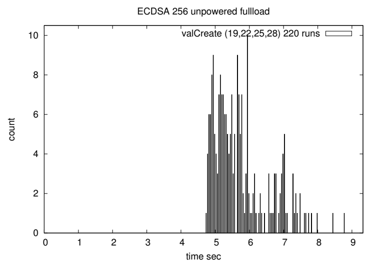
(d) ECDSA239, Key Pair Creation.



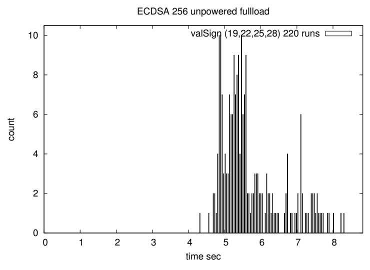
(e) ECDSA239, Signature Creation.



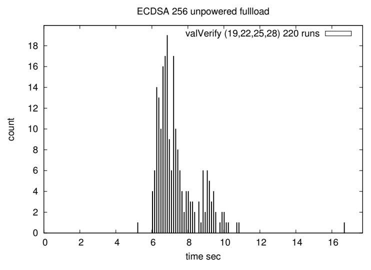
(f) ECDSA239, Signature Verification.



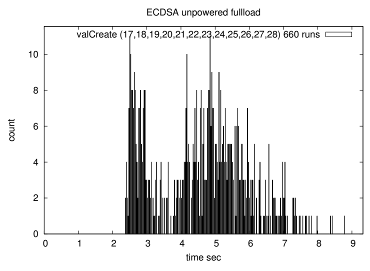
(g) ECDSA256, Key Pair Creation.



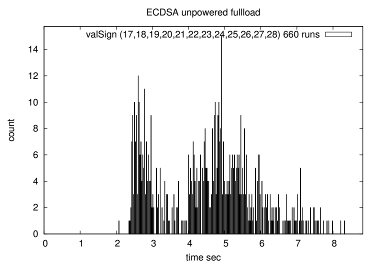
(h) ECDSA256, Signature Creation.



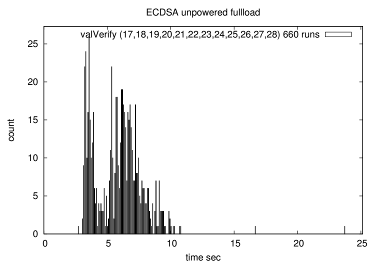
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

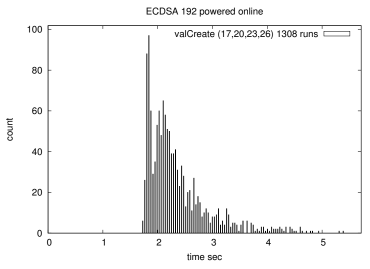


(k) ECDSA, Signature Creation.

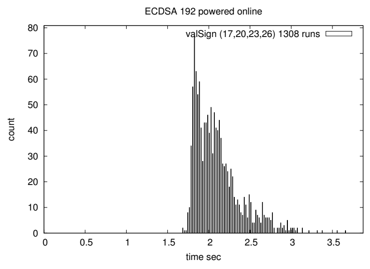


(l) ECDSA, Signature Verification.

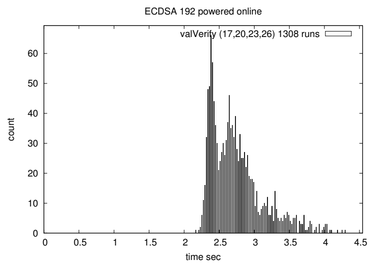
Fig. 58. 5800xm, Full-load Unpowered, ECDSA 192, 239, 256.



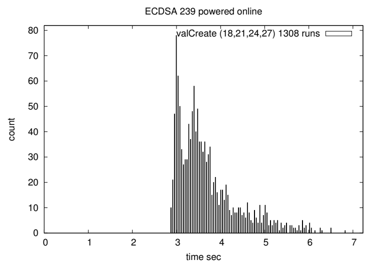
(a) ECDSA192, Key Pair Creation.



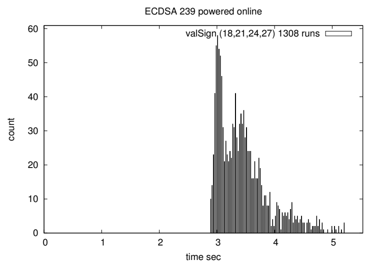
(b) ECDSA192, Signature Creation.



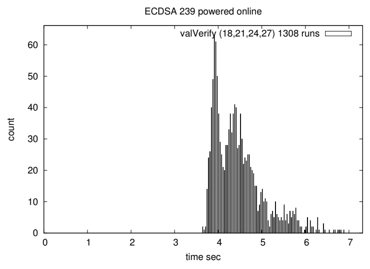
(c) ECDSA192, Signature Verification.



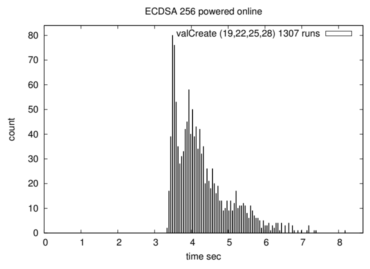
(d) ECDSA239, Key Pair Creation.



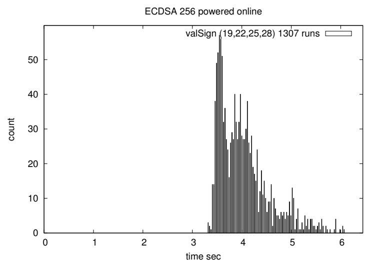
(e) ECDSA239, Signature Creation.



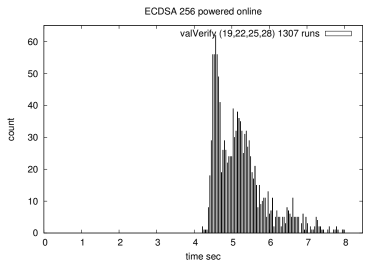
(f) ECDSA239, Signature Verification.



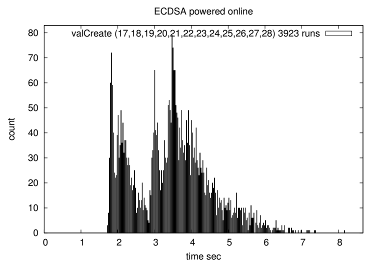
(g) ECDSA256, Key Pair Creation.



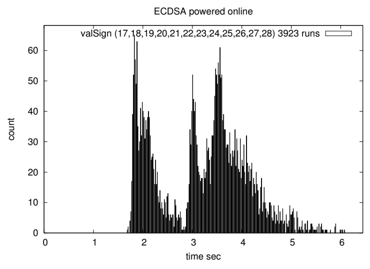
(h) ECDSA256, Signature Creation.



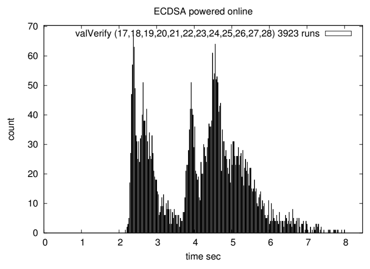
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

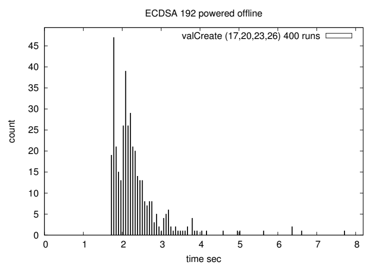


(k) ECDSA, Signature Creation.

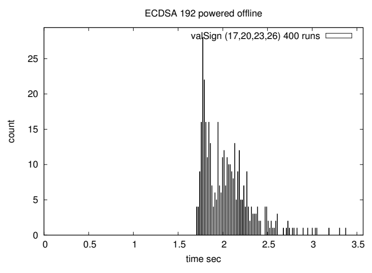


(l) ECDSA, Signature Verification.

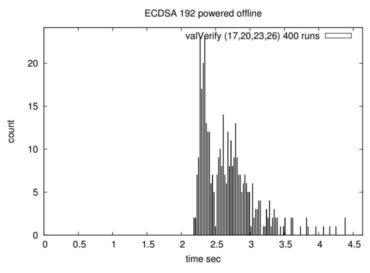
Fig. 59. E50, Online Powered, ECDSA 192, 239, 256.



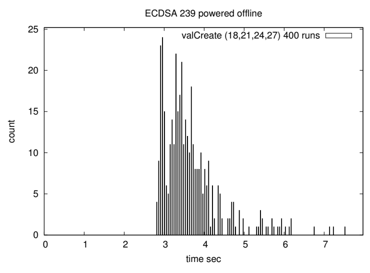
(a) ECDSA192, Key Pair Creation.



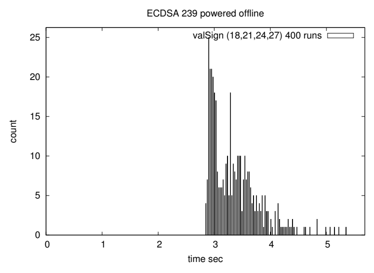
(b) ECDSA192, Signature Creation.



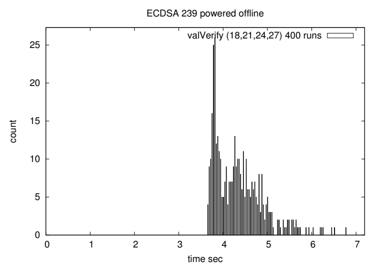
(c) ECDSA192, Signature Verification.



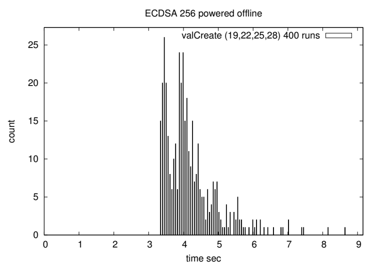
(d) ECDSA239, Key Pair Creation.



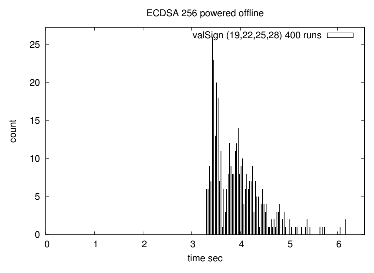
(e) ECDSA239, Signature Creation.



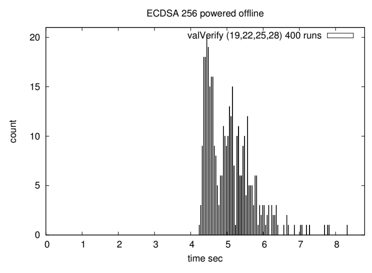
(f) ECDSA239, Signature Verification.



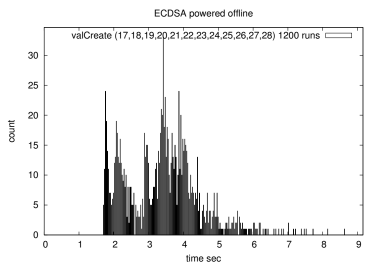
(g) ECDSA256, Key Pair Creation.



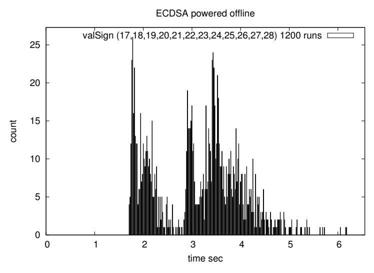
(h) ECDSA256, Signature Creation.



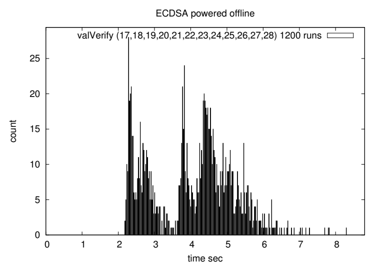
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

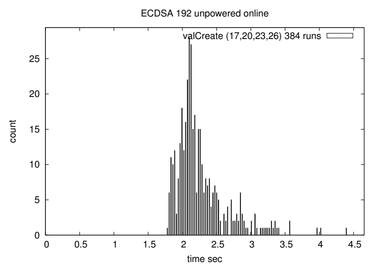


(k) ECDSA, Signature Creation.

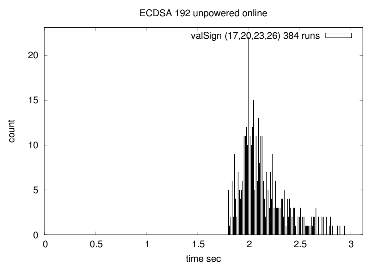


(l) ECDSA, Signature Verification.

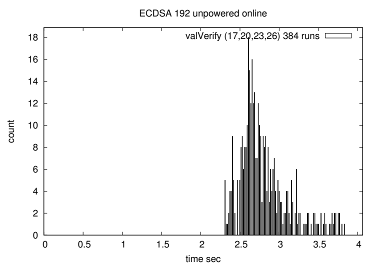
Fig. 60. E50, Offline Powered, ECDSA 192, 239, 256.



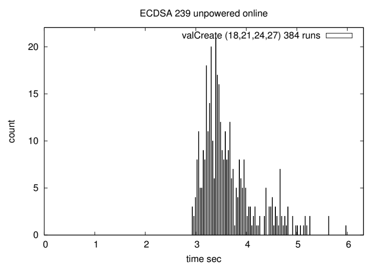
(a) ECDSA192, Key Pair Creation.



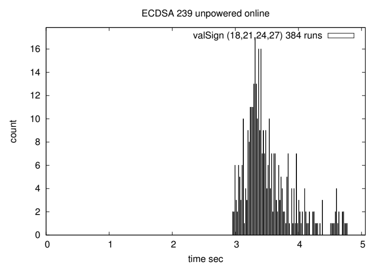
(b) ECDSA192, Signature Creation.



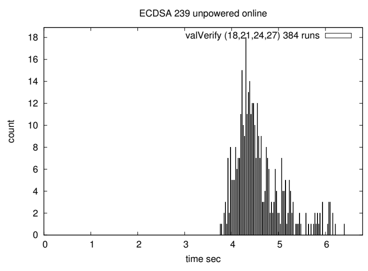
(c) ECDSA192, Signature Verification.



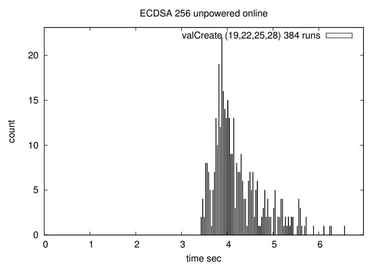
(d) ECDSA239, Key Pair Creation.



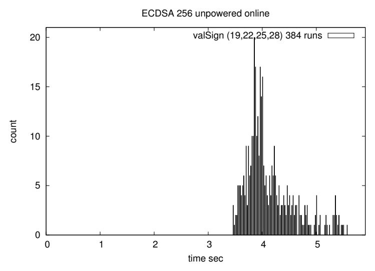
(e) ECDSA239, Signature Creation.



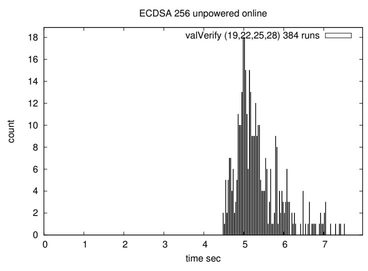
(f) ECDSA239, Signature Verification.



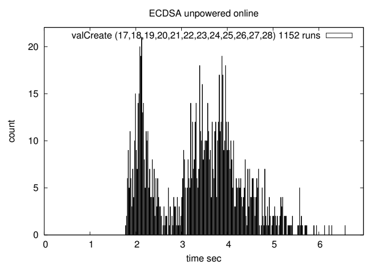
(g) ECDSA256, Key Pair Creation.



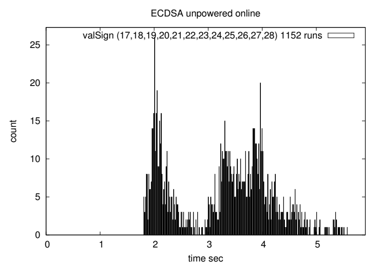
(h) ECDSA256, Signature Creation.



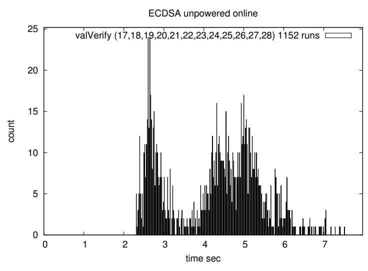
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

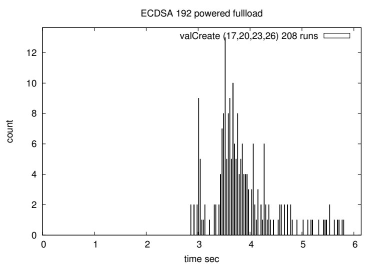


(k) ECDSA, Signature Creation.

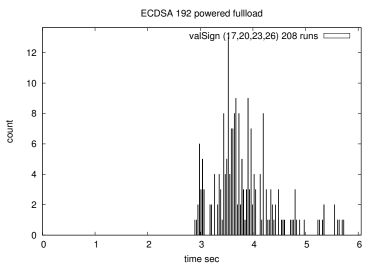


(l) ECDSA, Signature Verification.

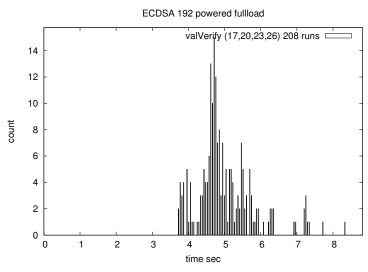
Fig. 61. E50, Online Unpowered, ECDSA 192, 239, 256.



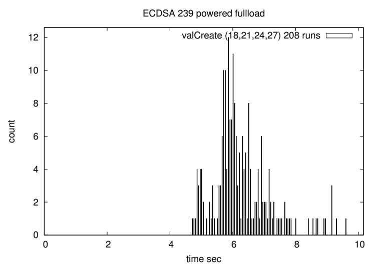
(a) ECDSA192, Key Pair Creation.



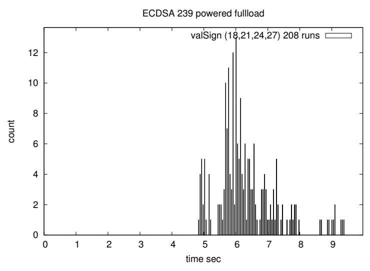
(b) ECDSA192, Signature Creation.



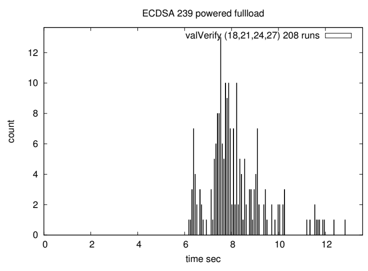
(c) ECDSA192, Signature Verification.



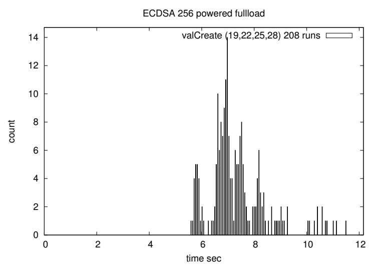
(d) ECDSA239, Key Pair Creation.



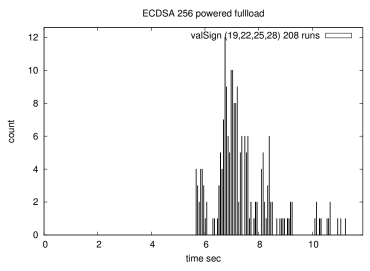
(e) ECDSA239, Signature Creation.



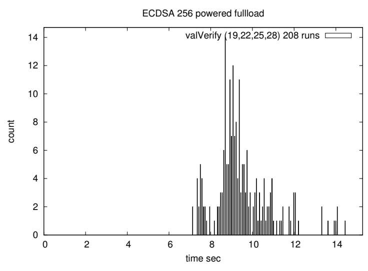
(f) ECDSA239, Signature Verification.



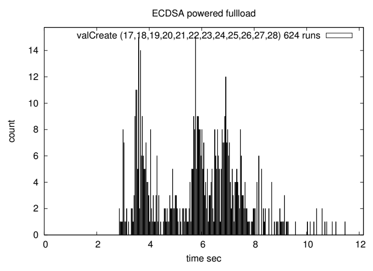
(g) ECDSA256, Key Pair Creation.



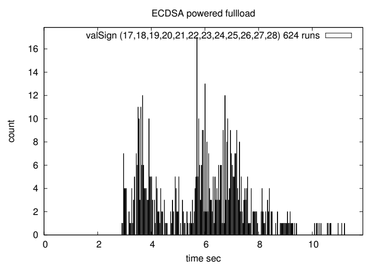
(h) ECDSA256, Signature Creation.



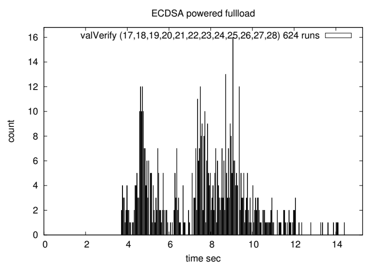
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

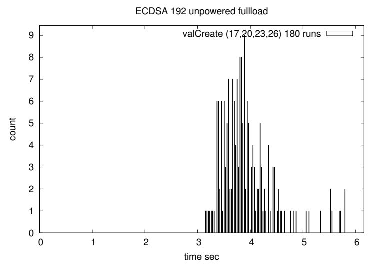


(k) ECDSA, Signature Creation.

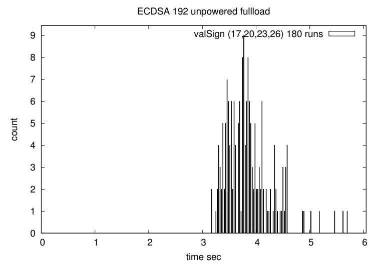


(l) ECDSA, Signature Verification.

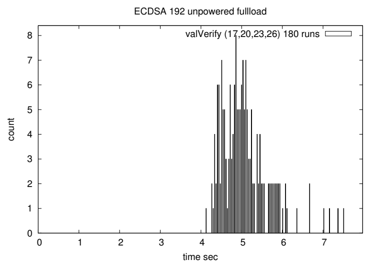
Fig. 62. E50, Full-load Powered, ECDSA 192, 239, 256.



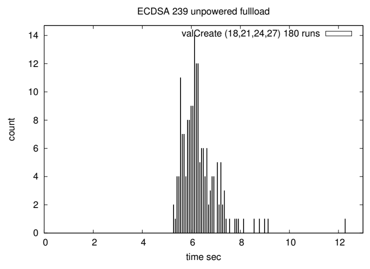
(a) ECDSA192, Key Pair Creation.



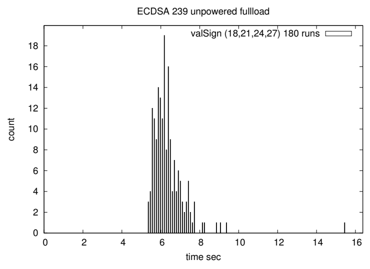
(b) ECDSA192, Signature Creation.



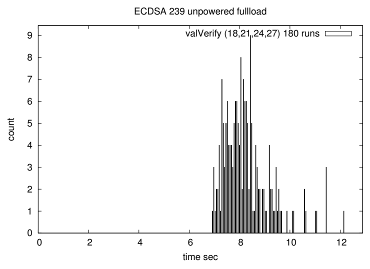
(c) ECDSA192, Signature Verification.



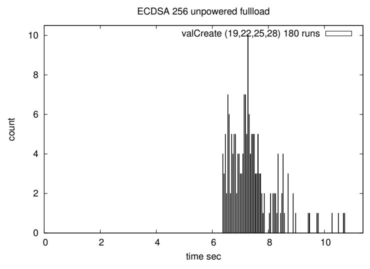
(d) ECDSA239, Key Pair Creation.



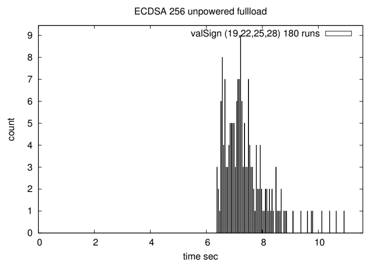
(e) ECDSA239, Signature Creation.



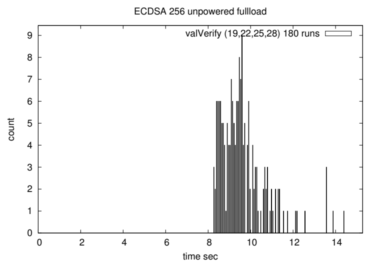
(f) ECDSA239, Signature Verification.



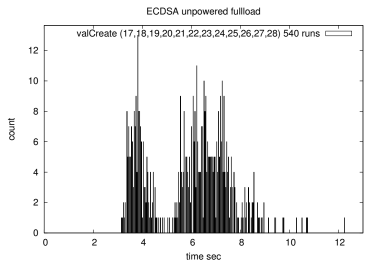
(g) ECDSA256, Key Pair Creation.



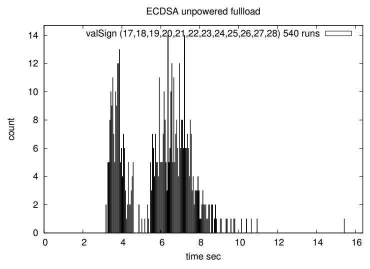
(h) ECDSA256, Signature Creation.



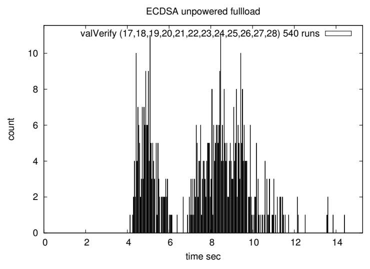
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.

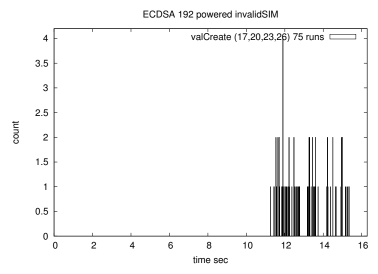


(k) ECDSA, Signature Creation.

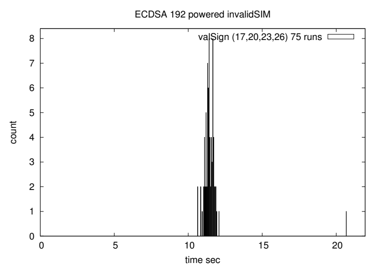


(l) ECDSA, Signature Verification.

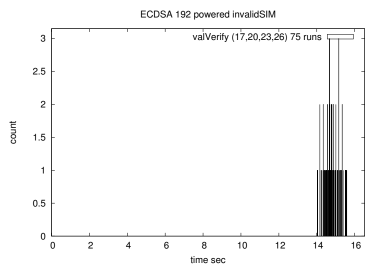
Fig. 63. E50, Full-load Unpowered, ECDSA 192, 239, 256.



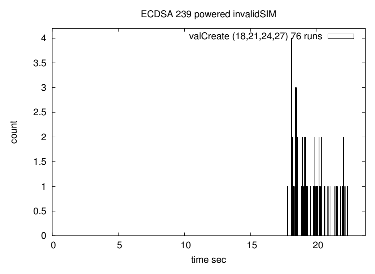
(a) ECDSA192, Key Pair Creation.



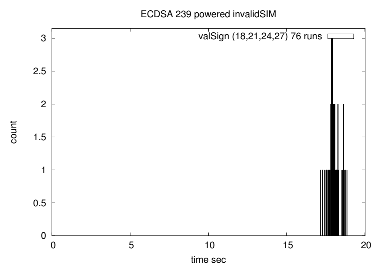
(b) ECDSA192, Signature Creation.



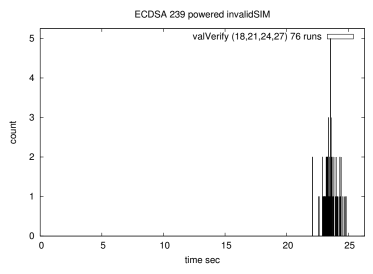
(c) ECDSA192, Signature Verification.



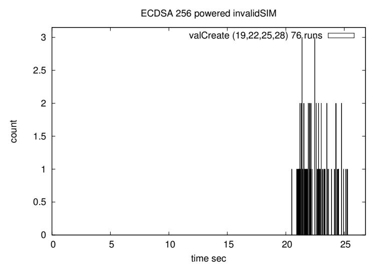
(d) ECDSA239, Key Pair Creation.



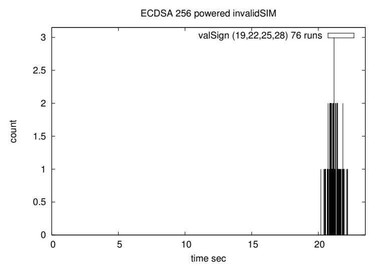
(e) ECDSA239, Signature Creation.



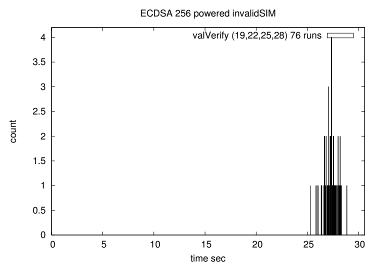
(f) ECDSA239, Signature Verification.



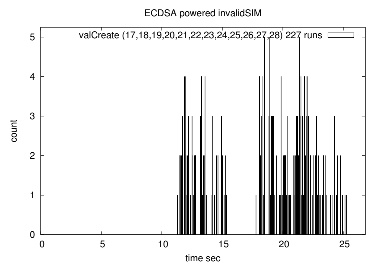
(g) ECDSA256, Key Pair Creation.



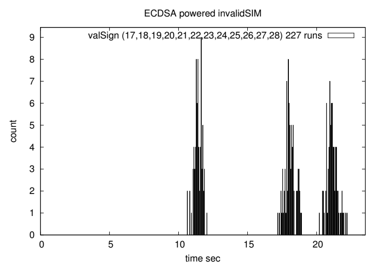
(h) ECDSA256, Signature Creation.



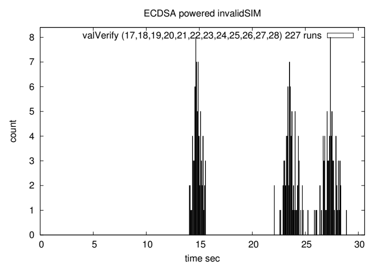
(i) ECDSA256, Signature Verification.



(j) ECDSA, Key Pair Creation.



(k) ECDSA, Signature Creation.



(l) ECDSA, Signature Verification.

Fig. 64. S65, InvalidSIM Powered, ECDSA 192, 239, 256.

REFERENCES

[1] M. Stini and M. Mauve, “Enabling Fair Offline Trading,” in *IWCMC 2009: Proceedings of the 5th International Conference on Wireless Communications and Mobile Computing*, June 2009.

[2] M. Stini, M. Mauve, and F. H. Fitzek, “Digital Ownership: From Content Consumers to Owners and Traders,” *IEEE MultiMedia*, vol. 13, no. 4, pp. 1,4–6, Oct. 2006.

[3] ——, “Digital Ownership for P2P Networks,” in *Mobile Phone Programming*, F. H. Fitzek and F. Reichert, Eds. Springer NL, 2007, pp. 259–270.

[4] D. Baselt, “Analysis and Implementation of Offline-Authentication on Mobile Devices ,” Bachelor’s thesis, Sept. 2006.

[5] M. Sliwak, “A Plattform for Transparent Messaging and Proximity Based Interest Sensing on Mobile Devices ,” Bachelor’s thesis, Mar. 2007.

[6] ITU-T, “Recommendation X.509,” http://www.itu.int/rec/T-REC-X.509/en (*as seen April 05th 2009*), 2005, x.509 : Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks.

[7] RSA Laboratories, http://www.rsasecurity.com/rsalabs/ (*as seen April 05th 2009*), 2009.

[8] L. of the Bouncy Castle, “Bouncy Castle Crypto API (*as seen April 05th 2009*),” http://www.bouncycastle.org/, 2009.

[9] Sun Microsystems, Inc., “The Java ME Platform,” http://rubyonrails.org.

[10] , “Ruby on Rails,” http://java.sun.com/javame.

[11] I. Sun Microsystems, “The Java ME Platform,” http://java.sun.com/javame/ (*as seen April 05th 2009*), 2009, The Java ME Platform - the Most Ubiquitous Application Platform for Mobile Devices .

[12] S. Li and J. Knudsen, *Beginning J2ME*, 3rd ed. New York: Springer-Verlag, 2005.

[13] I. Sun Microsystems, “Security and Trust Services API for J2ME (SATSA); JSR 177 (*as seen April 05th 2009*),” http://java.sun.com/products/satsa/, 2009.

[14] D. Hook, *Beginning Cryptography with Java*, 1st ed. Wiley, 2005.

[15] Cryptix, http://www.ntua.gr/cryptix/ (*as seen April 05th 2009*), 2009.

[16] T. Straub and A. Heinemann, “An Anonymous Bonus Point System For Mobile Commerce Based On WordOfMouth Recommendation (*as seen April 05th 2009*),” http://www.informatik.tu-darmstadt.de/GK/participants/tstraub/publications/straub_heinemann_acm-sac_2004.pdf, 2004.

[17] S. Inc., http://www.systemics.com/ (*as seen April 05th 2009*), 2001.

[18] T. Graz, “IAIK JCE,” http://jce.iaik.tugraz.at/ (*as seen April 05th 2009*), 2009.

[19] J. Buchmann, *Einführung in die Kryptographie*, 3rd ed. Springer, Berlin, 2003.

[20] A. Lenstra, X. Wang, and B. de Weger, “Colliding X.509 Certificates,” http://eprint.iacr.org/2005/067 (*as seen Sep 12th 2006*), 2005.

[21] NIST, “FIPS 180-2 Secure Hash Standard (*as seen Sep 12th 2006*),” http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf, 2002.

[22] *Lecture Notes in Computer ScienceEfficient Collision Search Attacks on SHA-0*. Heidelberg: Springer-Verlag, 2005, ch. Efficient Collision Search Attacks on SHA-0.

[23] B. Schneier, “New Cryptanalytic Results Against SHA-1,” http://www.schneier.com/blog/archives/2005/08/new_cryptanalyt.html (*as seen April 05th 2009*), 2005.

[24] H. Dobbertin, A. Bosselaer, and B. Preneel, “RIPEMD-160: A Strengthened Version of RIPEMD (*as seen Sep 12th 2006*),” http://homes.esat.kuleuven.be/˜cosicart/pdf/AB-9601/AB-9601.pdf, 1996.

[25] A. Bosselaer, “The RIPEMD-160 page (*as seen Sep 12th 2006*),” http://homes.esat.kuleuven.be/˜bosselae/ripemd160.html, 2004.

[26] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition*. Wiley, October 1995.

[27] J. Rothe, *Complexity Theory and Cryptology*, 1st ed. Heidelberg: Springer-Verlag, 2005.

[28] RSA Security, http://www.rsasecurity.com/ (*as seen April 05th 2009*), 2009.

[29] R. Housley, W. Polk, W. Ford, and D. Solo, “Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile,” RFC 3280 (Proposed Standard), Apr. 2002, obsoleted by RFC 5280, updated by RFCs 4325, 4630. [Online]. Available: http://www.ietf.org/rfc/rfc3280.txt

[30] NIST, “FIPS 186-2 Digital Signature Standard (DSS) (*as seen Sep 12th 2006*),” http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf, 2000.

[31] ——, “Recommended Elliptic Curves for Federal Government use (*as seen Sep 12th 2006*),” http://csrc.nist.gov/CryptoToolkit/dss/ecdsa/, 1999.

[32] J. Lopez and R. Dahab, “An Overview of Elliptic Curve Cryptography,” http://citeseer.ist.psu.edu/333066.html (*as seen Sep 12th 2006*), 2000.

[33] Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen, “Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung,” http://www.bundesnetzagentur.de/media/archive/15549.pdf (*as seen April 15th 2009*), 2009.

[34] Damien Giry, “Keylength - Cryptographic Key Length Recommendation,” http://www.keylength.com.

[35] BlueKrypt, “BlueKrypt - Conseil en Cryptographie et Sécurité Informatique,” http://www.bluekrypt.com.

[36] N. I. o. S. NIST and Technology, “Recommendation for Key Management, Special Publication 800-57 Part 1 (*as seen 15th April 2009*),” http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-57Part1_3-8-07.pdf, 2007.

[37] C. S. S. NSA/CSS, National Security Agency, “NSA Suite B Cryptography (*as seen 15th April 2009*),” http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml, 2008.

[38] E. N. o. E. i. C. IST-2002-507932 ECRYPT, “ D.SPA.28 Rev. 1.1, Yearly Report on Algorithms and Keysizes (2007-2008),” http://www.ecrypt.eu.org/ecrypt1/documents/D.SPA.28-1.1.pdf (*as seen April 16th 2009*), 2008.

[39] B. Weis, “The use of rsa/sha-1 signatures within encapsulating security payload (esp) and authentication header (ah),” RFC 4359 (Proposed Standard), Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4359.txt

[40] I. M. S. WG, “Msec status page, rfc4359,” http://tools.ietf.org/wg/msec/draft-ietf-msec-ipsec-signatures, 2008.