Secure Server-Aided Signature Generation

Markus Jakobsson and Susanne Wetzel

Bell Laboratories, Lucent Technologies
Information Sciences Research Center
Murray Hill, NJ 07974, USA
{markusj,sgwetzel}@research.bell-labs.com

Abstract. We study how to reduce the local computational cost associated with performing exponentiation. This involves transforming a large computational task into a large set of small computational tasks that are to be performed by a set of external servers who may all be controlled by one and the same adversary. In order to attack our problem, we introduce and employ the three principles of duplication, distribution and delegation. We apply our exponentiation scheme to performing inexpensive server-aided batch signature generation, and show noticeable efficiency improvements for batches of size 20 and up.

Keywords: Batch, DSS, delegation, duplication, error-detection.

1 Introduction

It is unfortunate that in spite of increasing processor speeds, the most important design criteria for many security protocols remains the computational complexity of the protocols, and not their security. To some extent, this is a mere reflection of human greed, as the relationship between efficiency and cost is well understood, whereas the impact of security problems is harder to quantify. With this in mind, it is clear that the development and understanding of efficient cryptographic building blocks is a vital contribution to protocol security, as only efficient tools will be employed for all but a few applications. On one hand, this prompts research into the development and usage of symmetric key cryptography; on the other hand, it suggests efforts to provide improved techniques for versatile but expensive operations, such as those relating to public key cryptography. In this paper, we focus on the latter.

One of the most common – and expensive – operations in public key cryptography is that of exponentiating. Its costs hover around 200 modular multiplications per exponentiation for exponent sizes around 160 bits, using standard window based methods. Clearly, any advance that allows its cost to be reduced is valuable. For very small batches, currently known techniques do not give any noteworthy speedups, while for large batches, various amortization techniques are known. An example of an amortization method is addition chains [6,5], which to day is the most efficient way of performing exponentiation for large batches, with a cost around a mere 10 multiplications per item [5]. The methods suggested in this paper allow for a local reduction of computational costs for both small

K. Kim (Ed.): PKC 2001, LNCS 1992, pp. 383–401, 2001.

batches (compared to window based methods) and large batches (compared to addition chain methods.)

In the above, the importance of the word *local* is imperative. Namely, we take the approach (so commonly used in most hierarchical organizations) of *delegation*: Instead of *performing* the task in question, it is handed off to somebody else. Whereas this can safely be done for most everyday life tasks (such as making coffee), one has to be very careful about how it is performed for tasks that involve security. In particular, if an enemy server learns an exponent used in a signature, and later intercepts the signature, he may be able to determine the secret key of the signer. A second problem is that of robustness, i.e., making sure that the computation performed is correct. The methods introduced in this paper are aimed at overcoming these and related problems, and thereby allowing the safe and robust delegation of exponentiation work to a set of untrusted servers. It is important to notice here that the security does not rely on sending different portions of a query to different servers that are assumed not to communicate. (In contrast to some work on how to compute with oracles.) Rather, we assume that *all* the delegated work may be performed or seen by one and the same malicious party. We argue security of our solution in this setting.

Our methods, which constitute a new approach to an old problem, involve as a first step some amount of pre-processing by the server wishing to have the computation to be performed. Let us call this party the *originator*. As a second step, they involve some computation by untrusted *external servers*, and finally, some post-processing by the originator. Generally speaking, the significance of the first step is to appropriately blind the request and to insert sufficient redundancy in the request, so that in the third step, errors and other deviations can be detected. Note, however, that standard methods for blinding cannot be used, as they involve the use of exponentiation (which, of course, is what we want to perform in the first place.) A contribution of this paper is therefore the development of inexpensive methods for blinding batches of queries. A second contribution is the development of methods that inexpensively allow for error detection and error correction in settings like ours. However, whereas these may be contributions of independent interest, we feel that the main importance is the move towards lowering exponentiation costs, along with the introduction of the idea of delegation to the setting considered.

While most batch methods require large batches, ours does well for small ones as well, which is particularly useful in a setting involving computationally limited devices such as smartcards.

For batches of size 20, we achieve local computational savings of approximately 81% over window-based methods, while batches of 100000 give us savings of 20%. When we talk about small batches, such as batches of size 20, we assume the use of a larger *cumulative* batch size, where the cumulative batch size corresponds to the number of elements to be processed over the lifetime of the device, as opposed to the number processed at one occasion. This is important to note as it allows for preprocessing corresponding to the cumulative batch size, which is performed externally, thus resulting in further savings.

Outline: We begin by discussing the related work in section 2. We then introduce the model and define our goals in section 3. This is followed by a high-level discussion of our methods in section 4, after which we delve into our solution (tailored for large batches) in section 5. In section 5.4 we make security claims corresponding to this scenario, with a corresponding security and efficiency analysis in the appendix. In section 6 we briefly discuss a variant of our solution applied to a smartcard scenario (corresponding to reduced quantities of local storage and use of small batches.)

2 Related Work

The general ideas of server-aided computation and batching are not new (c.f. [1,3,9,21,24,25,29]). While the server-aided RSA computation protocols proposed in [9,24,25] have been found not to be secure [28,22], the methods introduced in [29,21] are believed to be secure against passive attacks, i.e., attacks performed by honest-but-curious servers.

Our work differs from the above in several ways. First, while most of the previous work focuses on modular exponentiation for composite moduli (i.e., RSA), we target computation with prime moduli, and in particular, discrete log protocols like DSA [26] and Schnorr [32]. A second difference is that we make more pessimistic trust assumptions, and introduce methods for achieving robustness under such assumptions, and under active attacks. In this sense, we position ourselves much closer to work on oracle computation [1], which can be seen as a theoretical relative of server-aided computation, and where robustness is a core issue.

However, we make computational assumptions that are closer related to those typically made for server-aided computation, as we adopt the setting of a polynomial-time limited adversary. This makes particular sense in the context of generating standard signatures, which in turn is one of the focal points of our work. Therefore, on the one hand, we make stronger security assumptions than what is done for much of the work relating to computing with oracles, which allows us more efficient results. On the other hand, some of our other assumptions are *weaker* (and more realistic, we believe), as we allow the adversary to see the *entire* query string, independently of whether one or more servers are employed in practice. This is in contrast to what is assumed for some of the work on computing with oracles (e.g., [2,16]), where it is assumed that there is a separation between servers.

Our work is related to [7] in terms of the goal it achieves. However, a noteworthy difference is that the exponents for which computation is performed are random (according to some near-random diatribution) in [7], while they can be chosen arbitrarily in our scheme.

Technically, our work is influenced by a whole array of well understood methods. Whereas the type of blinding we perform is novel, it is related in spirit to existing blinding techniques performed for discrete log based signature schemes, such as was done in [8]. While the solutions in [9,25,24] are vulnerable to active and passive attacks [22,28], we introduce various error detecting techniques to achieve robustness. These techniques, which depend closely on the use of blinding, also draw on several previously proposed methods. One method is to duplicate the set of queries, and to permute and blind each item individually, forcing an adversary to guess the location of related portions in order to successfully attack the system. This idea has previously been employed for robustness of mix networks [20,15]. Another method is to introduce known values in the query string, and verify that these are computed correctly. Together with the permutation performed, this becomes an inexpensive and powerful way of protecting against an attack in which the adversary offsets all the queries in the same manner (which in principle would be possible were only the duplication method to be used.) A third method, of a more traditional error-detecting type, is to introduce values that depend on other values, and to verify that these values, when returned, have the expected relationship. A drawback of this approach is the increased cost per item; however, it could be a useful method in a setting with very small batches (in which case the duplication methods becomes less powerful.) Finally, a fourth method is to introduce dependencies in how the computation is performed. In other words, this chains the partial results in a manner that spreads any inconsistency, thereby making it easier to detect. (While this is related in spirit to the avalanche effect used in symmetric cipher design, the actual methods employed differ substantially.) This method becomes a powerful (but also rather expensive) error-detecting tool in conjunction with the previously outlined methods. However, the error-spreading functionality not only allows easy detection of errors, but has the flip side of spreading the effects of any error to otherwise good portions of computation.

3 Model and Requirements

Participants. We assume all participants to be modeled by polynomial-time limited Turing machines. The scheme has two types of "desired" participants: an *originator* (who in our signature setting is the party who holds the secret key for generating signatures) and *external servers*. The latter are used by the originator to have computation performed.

We assume that all the external servers may be controlled by an "unwanted" entity, the adversary. It is the primary goal of the adversary to improve his chances of computing some known function of the secret signing key of the originator (such as forging a signature.) A secondary goal is to corrupt the computation the originator wishes to perform, without this being noticed by the the originator.

Informally, a protocol for outsourcing computation should be *private* (not leak secret information) and *robust* (not allow incorrect computation to go undetected). Moreover, it has to be *efficient*, i.e., reduce the amount of local computation to be performed given some assumptions on the probability that a computational portion is correctly performed when outsourced. More formally, we can define these as follows:

Definition: *Let T be a computational task, and f an arbitrary function. We say that a delegation of T is ϵ-private with respect to f if the adversary has only a negligible advantage ϵ in computing $f(i)$ for some input i if performing the delagated work* and *seeing the public input and output of the originator, compared to a setting where he only sees the public input and output of the originator.*

It has to be noted that in the setting we study, we are – in terms of privacy – only directly concerned with the privacy of the exponent values for the signatures (which indirectly corresponds to the privacy of the secret signing key). If this context it may be more appropriate to denote the property *security*, although we will use the more general (and generally applicable) *privacy*.

The robustness of the scheme deals with a different kind of attack, namely the adversary trying to corrupt the computations. In the following, we consider the robustness of a signature generation:

Definition: *We say that a delegation of T is ϵ-robust if an adversary who controls all the external servers performing computation for the signer cannot corrupt the computation but for a probability ϵ. (The above is over all random strings of the signer, and over all computational tasks T.)*

Turning to efficiency, note that we are primarily concerned with the amount of computation performed by the signer, and not with that performed by the external servers (as long as their computational tasks are feasible.) We define the efficiency of an outsourced computation as follows:

Definition: *We say that a delegation of a computational task T is (ϵ, ν)-efficient, if the signer's computational load if performing the computation T himself is a fraction ϵ of that required by him if outsourcing T. This is relative to a certain fraction ν of incorrect responses that are scheduled by the adversary, and where the probability is over all random coins of the signer.*

In the following we present a solution which allows the efficient delegation of signature generation. This proposed solution meets the requirements of robustness and privacy, as will be shown in the analysis section.

4 Delegation

The type of work we strive to delegate in this paper, namely generating DSA signatures [26], is characterized by a large amount of exponentiation. We will use the standard denotation for DSA, in which one of the computational tasks is to compute $r = g^k \bmod p$, where $k \in Z_q$, primes p and q such that $p = lq + 1$, and $|p| = 1024$, $|q| = 160$. As usual, all operations are assumed to be performed modulo p, where applicable, unless otherwise noted.

Clearly, it is easy to delegate the exponentiation of values simply by transmitting the bases, exponents, and moduli to a server performing the computation. However, this simplistic approach has several potential weaknesses; in the following, we will discuss these, along with the constructions we will employ to avoid them.

The issues of robustness and defense against information leaks are closely related in that the methods to achieve these two goals are largely overlapping. Let us consider what these methods are:

1. *Permutation.* A first method is to randomly permute the order of the partial tasks in the batch of requests sent to the external servers. Permutation is helpful as it forces the adversary to guess what exponentiation(s) corresponds to what signature. The costs incurred by the originator to perform this operation are insignificant. Similarly, the permutation does not affect the amount of computation to be performed by the external servers.
2. *Blinding.* A second method we use is to transform the exponents in a way that closely corresponds to a traditional blinding. This is done by applying a random (and secret) offset to each exponentiation in the batch where offsets are selected in a particular way to keep the costs of the operation down. (We elaborate on how this is done below.) The additional local cost incurred by blinding consists of a first cost corresponding to applying the offsets to the exponents, and a second cost of removing the resulting offsets from the values returned by the external servers. The blinding does not affect the global costs, i.e., the amount of computations performed by the external servers, at all.
3. *Error-detection and error-correction.* Finally, a third method we use (and which to some extent is a novel method for this type of application) employs error detection and error correction methods to spot and remedy inconsistencies. This is done by introducing what we call *known values*, *replication*, *dependencies* and *checksums*. These methods manipulate the vector of queries and verify that certain relationships hold for the returned results. The methods are as follows:
   (a) *Known values.* We let the originator insert some w tasks into the batch of computations to be delegated, where he already knows the results of these computational tasks. (For example, $x = 0$, resulting in $g^x = 1$; note that such simple computational tasks can not be distinguished from other tasks once they are blinded.) This allows the detection of so-called offset attacks, in which the adversary correctly computes all tasks, and then offsets all of the replies using the same multiplicative offset. Both the additional local and global costs are negligible.
   (b) *Replication.* A second method for error-detection is replication, according to which one instead of delegating a computational task only once delegates it some τ times. We will apply this transformation before blinding and permutation are performed. Since each task is delegated τ times, both local and global costs increase by a factor τ (not considering minor amortization gains.)

(c) *Dependency.* Including dependencies in the computations performed by the clients results in serious error propagation, which reduces the success probability of attacks in which an adversary provides some incorrect replies. Together with duplication, this allows easy detection of errors and malicious responses. We implement dependencies by "linking" tasks to each other, making one result depend on two or more other results, which in turn depend on others. We introduce dependencies by transforming a query consisting of the exponents $k_1, \ldots, k_n$ to a query consisting of the the exponents $k'_1, \ldots, k'_n$, where

$$k'_i = \begin{cases} k_1 & : i = 1 \\ k_i + \alpha \cdot k_{i-1} + \beta \cdot k'_{i-1} \bmod q & : 1 < i \leq n \end{cases}$$

where $\alpha, \beta \in \{-1, 0, 1\}$. Depending on the values of α and β, different levels of error propagation can be achieved. While error propagation is very useful for detecting forgery, it also requires recomputation of many exponentiations. For efficiency reasons, this method needs to be combined with the other mechanisms.

(d) *Checksums.* A fourth method is that of inserting checksums. These are values that depend on subsets of the other values (potentially all of these), and which are checked by multiplying the results together and comparing to the corresponding checksum values. Whereas it is an expensive operation to verify the checksums (requiring roughly one multiplication per item selected for the checksum) it can be used as a recovery method to locate good sections of a corrupted set of computed values. (Where we use other methods to *detect* errors.) This may be beneficial given the negligible precomputation costs of the operation. The operation results in a minor blow-up in the size of the query string; namely, introducing u checksums each of length v with $u \cdot v = n$ (i.e., each element is part of one checksum) results in u additional computational tasks to be delegated to the servers.

It is important to note that in order to achieve maximum security, one should perform the error detection and error correction before performing the blinding and permutation steps.

Detailed Blinding Description. In detail, the blinding for the exponent vector $(k_1, \ldots, k_n)$ is done by first choosing e random numbers $r_1, \ldots r_e \in \{0, \ldots, \frac{q-1}{2}\}$. Then, for each exponent k_j with $1 \leq j \leq n$, d elements are chosen and the new exponents are computed as

$$k'_j = k_j - \sum_{i=1}^{e} \gamma_{i,j} r_i \bmod q$$

with $\gamma_{i,j} \in \{0, 1\}$ and $\sum_{i=1}^{e} \gamma_{i,j} = d$. The choice of what blinding values to combine in order to obtain the various blinding elements, we must not select two equal sets of blinding values, or they can be cancelled by randomly guessing these two resulting blinded portions, which could leak the secret key. In fact, we select what blinding values to use by enumeration over all sets with a particular minimum Hamming distance (where the Hamming distance of two sets $\mathcal{S}_1, \mathcal{S}_2$ is defined as the number of elements of $(\mathcal{S}_1 \cup \mathcal{S}_2) \backslash (\mathcal{S}_1 \cap \mathcal{S}_2)$). The Hamming distance in turn determines how many portions have to be combined by the adversary in order to cancel blinding factors. The sets of blinding factors form a so-called constant weight code with length n and weight d. We refer to [12,30] for a description of these codes, their properties and how they can be constructed. Computing the actual signatures corresponding to the exponents $k_1, \ldots, k_n$ requires the computation of the $g^{\sum_{i=1}^{e} \gamma_{i,j} r_i \bmod q}$ for $1 \leq j \leq n$ by the signer. Using standard methods, the g^{r_i} for $1 \leq i \leq e$ can be computed with ≈ 200 multiplications and the precomputation of all possible pairs $g^{r_i + r_j \bmod q}$ with $1 \leq i < j \leq e$ requires $\leq \frac{e(e-1)}{2}$ multiplications. Thus, the additional costs for the signer computing the original signature amount to

$$\approx 1 + (\lceil \frac{d}{2} \rceil - 1) + \frac{1}{n}(200e + \frac{e(e-1)}{2}).$$

Remark on dependencies: In a more general setting, the dependencies can be introduced on blocks of size $b \leq n$ where $b \cdot b' = n$ as follows:

$$k'_i = \begin{cases} k_i & : i = l \cdot b + 1 \\ k_i + \alpha \cdot k_{i-1} + \beta \cdot k'_{i-1} \bmod q & : l \cdot b + 1 < i \leq (l+1) \cdot b \end{cases}$$

for $1 \leq l \leq b'$.

5 Solution (Large Batches)

We are now ready to present our solution. At this level of description, we are only concerned with the computation performed on the signer's side. This is broken into two portions, sandwiching the portion performed by the computing servers. These two portions correspond to transforming the problem into a randomized description of the same, and to transform back the corresponding "randomized result" into a result, and check for inconsistencies. The parameters, e.g., the number of signatures to be produced, have been chosen to balance efficiency and security requirements for an Internet scenario (characterized by large batches.)

5.1 Problem Transformation

Let the input consist of the vector $((g, k_1), \ldots, (g, k_n))$ corresponding to an implicit request to compute $(g^{k_1}, \ldots, g^{k_n})$. We denote this input by $\mathcal{G}_1 = (k_1, \ldots, k_n)$. The signer will transform $\mathcal{G}_1$ as follows:

1. **Replication** At first, the above vector is extended by replicating the last element of $\mathcal{G}_1$, i.e., $k_{n+1} = k_n$. Then, the resulting is substituted with a

vector where each element occurs three times. I.e., the original vector $\mathcal{G}_1 = (k_1, \ldots, k_n)$ is transformed into the new vector

$$\mathcal{G}_2 = (k_1, \ldots k_n, k_{n+1}, k_1, \ldots \\ \ldots, k_n, k_{n+1}, k_1, \ldots, k_n, k_{n+1}).$$

2. **Dependency.** As a next step, dependency is introduced by transforming the third part of the vector $\mathcal{G}_2$ yielding

$$\mathcal{G}_3 = (K_1, \ldots, K_{3n+3})$$

where $K_i = (\mathcal{G}_2)_i$ for $1 \leq i \leq 2n+2$ and for $2n+3 \leq i \leq 3n+3$ the K_i are inductively defined as

$$K_i = \begin{cases} k_1 & : i = 2n+3 \\ k_i - k_{i-1} - K_{i-1} \bmod q & : 2n+3 < i \leq 3n+2 \\ K_{i-1} & : i = 3n+3 \end{cases}$$

Due that the fact that the dependencies can also be interpreted as checksums, no additional checksums are introduced.

3. **Blinding.** For the blinding, e random numbers $r_1, \ldots, r_e \in \{0, \ldots, \frac{q-1}{2}\}$ are picked. Then, for each element of the vector $\mathcal{G}_3$, 4 elements $\rho_{1,(\mathcal{G}_3)_i}, \ldots, \rho_{4,(\mathcal{G}_3)_i}$ are selected (in the manner previously described) from $R = \{r_1, \ldots, r_e\}$ and the new vector

$$\mathcal{G}_4 = (\kappa_1, \ldots, \kappa_{3n+3})$$

is computed as $\kappa_i = (\mathcal{G}_3)_i - \sum_{j=1}^{4} \rho_{j,(\mathcal{G}_3)_i} \bmod q$ for $1 \leq i \leq 3n+3$.

4. **Random permutation.** A permutation, Π on the vector is selected uniformly at random, resulting in the new vector

$$\mathcal{G}_5 = \Pi(\mathcal{G}_4).$$

5.2 Outsourcing

The vector that constitutes the final output of the above transformation is broken up into blocks of appropriate size, and communicated to computing servers. (We note that one clearly only has to communicate the value g once.) If a vector $(A_1, \ldots, A_k)$ is sent to a computing server, the latter is expected to compute and return the vector $(g^{A_1}, \ldots, g^{A_k})$.

5.3 Result Transformation

Again, we assume the input consists of one vector $\mathcal{G}_6$ whose elements consist of the values returned by the computing servers, arranged in the order in which they were handed out (that is, so that the reply to a query is entered in the same position from which the query was taken). The following steps are performed to transform and verify the result:

1. **Inverse permutation.** The signer constructs a new vector by applying the inverse permutation, compared to the one performed towards the end of the transformation stage. This results in the vector
$$\mathcal{G}_7 = \Pi^{-1}(\mathcal{G}_6).$$
2. **Inverse blinding.** For each $1 \leq i \leq 3n+3$ the signer computes
$$(\mathcal{G}_8)_i = g^{(\mathcal{G}_7)_i + \sum_{j=1}^{4} \rho_{j,(\mathcal{G}_3)_i} \bmod q}$$
thus resulting in $\mathcal{G}_8$. This computation is performed using methods for addition chains.
3. **Verification of dependencies and redundancy.** As a last step, the redundancies are checked, i.e., if $(\mathcal{G}_8)_{n+2} \overset{?}{=} (\mathcal{G}_8)_{2n+3}$, $(\mathcal{G}_8)_n \overset{?}{=} (\mathcal{G}_8)_{n+1}$, $(\mathcal{G}_8)_{2n+1} \overset{?}{=} (\mathcal{G}_8)_{2n+2}$ and $(\mathcal{G}_8)_{3n+2} \overset{?}{=} (\mathcal{G}_8)_{3n+3}$, as well as if for $1 \leq i \leq n$
$$(\mathcal{G}_8)_{n+1+i} \overset{?}{=} (\mathcal{G}_8)_i.$$
Moreover, for $2 \leq i \leq n$ the signer checks inductively
$$(\mathcal{G}_8)_{2n+2+i} \cdot (\mathcal{G}_8)_{2n+i+1} \cdot (\mathcal{G}_8)_{i-1} \overset{?}{=} (\mathcal{G}_8)_i.$$
If so, $(\mathcal{G}_8)_i$ with $1 \leq i \leq n$ are the correct results of the delegated computations. Otherwise, if $1 \leq j \leq n$ is the index where the check fails, then the computations of $(\mathcal{G}_8)_i$ with $1 \leq i < j$ are correct. The values $(\mathcal{G}_8)_i$ with $i < j < n$ are compared with $(\mathcal{G}_8)_{n+i+1}$. If equality holds, these values are assumed to be correct. Otherwise, recomputation will be necessary as in case of $(\mathcal{G}_8)_j$.

Remark: The above protocol obtains a high degree of robustness (as will be discussed in the following subsection and argued in the appendix). However, in situations in which robustness is not critical, or in which a third party is employed to verify the signatures, our protocol can be altered to substantially reduce the costs. Similarly, if our methods are employed for purposes of decryption instead of signature generation, redundancy checks (of the resulting plaintexts) can be used to obtain robustness at reduced costs.

The protocol costs can be reduced by not performing the verification of dependencies (reducing the amount of work performed in steps 2 and 3 above), or by not using duplication (reducing the amount of work performed in step 2). Note that these changes do not alter the degree of privacy, but only the robustness. We will review the resulting costs and robustness claims of the corresponding protocol versions.

5.4 Security Claims

For concreteness, we analyze the security of our scheme only for the particular parameters proposed.

Privacy. Starting with the privacy aspect, and considering the state of the art in terms of attacks, we argue in Appendix A that if we choose at least $e = 75$ blinding factors, then our scheme is 2^{-80}-private for input sizes larger than 100000 and smaller than $\binom{e}{4} = \binom{75}{4} = 1215450$.

Robustness. For an input size n, our protocol is $(3 \cdot \binom{3n+2}{5})^{-1}$-robust against an adversary who controls all the external servers, for large batch sizes. Thus, for input sizes larger than 47000 elements, an attacker has less than a probability 2^{-80} to corrupt the computation without detection.

In Appendix B, we argue by case analysis that it is necessary for an adversary to select a "coherent set" of at least six elements among the existing $3n + 3$ elements. Given the random distribution of the elements (described in appendix A) the adversary can only succeed if the five last elements he chooses are "coherent with" the first elements he chose, which gives the claimed probability of success.

Efficiency. In Appendix C, we show that our protocol is approximately $(\frac{5}{4}, 0)$-efficient for batches of size approximately 100000 signatures, with an actual cost per signature of 8 multiplications. This corresponds to a local efficiency improvement of 20%, compared to addition chain methods.

6 Smartcard Setting (Small Batches)

In our smartcard setting, we perform the initial exponentiation as part of the manufacturing or initialization process of the smartcard. (However, we do not precompute all pairs, due to the storage problems that would cause.) Thus, the smartcard stores pairs (r_i, g^{r_i}). While in this smartcard setting the original vector has to be replicated seven times in order to obtain a robustness level corresponding to a probability of failure of 2^{-80} for a batch size of $n = 20$, the dependency is introduced as before. In order to achieve a security level of 2^{-80}, the cumulative batch size will have to be chosen to be at least 80000. Therefore, 75 blinding factors is sufficient, since $\binom{75}{4} > 80000 \cdot 9$, where 9 is the effect of replication and dependencies. We note that the cumulative batch size corresponds to the number of signatures that can be generated by the smartcard without "recharging".

We see that the local cost per signature is only 42 multiplications per signature, which is in strong contrast to the approximately 200 multiplications that are otherwise needed for such small batches. Thus, our protocol is $(\frac{100}{19}, 0)$-efficient for very small batches, which corresponds to a 81% savings.

7 Conclusions

In this paper we have presented methods to reduce the local computational costs associated with performing exponentiations for signature generation. Using the principles of duplication, distribution and delegation, we achieve a reduction of costs incurred by the use of addition chains which to day is the most efficient method of performing exponentiations. A future direction of interest is to apply our methods to different problems and to research other redundancy methods to obtain robustness. In particular, employing different patterns for the recurrence computation may be of benefit. Furthermore, application of results known from coding theory could give improved methods for selection of blinding values which in turn would improve efficiency by allowing smaller batches and cumulative batches.

References

1. M. Abadi, J. Feigenbaum and J. Kilian, “On Hiding Information from an Oracle”, Journal of Computer and System Sciences, Vol. 39, no. 1, Aug 1989, pp. 21–50.
2. D. Beaver, J. Feigenbaum and V. Shoup, “Hiding Instances in Zero-Knowledge Proof Systems,” CRYPTO ’90, pp. 326–338.
3. M. Bellare, J.A. Garay and T. Rabin, “Fast Batch Verification for Modular Exponentiation and Digital Signatures,” EUROCRYPT ’98, 1998, pp. 236–250.
4. M. Ben-Or, S. Goldwasser and A. Wigderson, “Completeness Theorems for Noncryptographic Fault-Tolerant Distributed Computations”, STOC ’88, pp. 1–10.
5. D. Bleichenbacher, “Addition Chains for Large Sets”, preprint, 1999.
6. J. Bos and M. Coster, “Addition Chain Heuristics,” CRYPTO ’89, pp. 400–407.
7. V. Boyko, M. Peinado and R. Venkatesan, “Speeding up discrete log and factoring based schemes via precomputations. EUROCRYPT ’98, pp. 221–235.
8. S. Brands, “Untraceable Off-line Cash in Wallet with Observers,” CRYPTO ’93, pp. 302–318.
9. J. Burns and C.J. Mitchell, “Parameter Selection for Server-Aided RSA Computation Schemes” IEEE Transactions on Computers, Vol. 43, No. 2, 1994, pp. 163–174.
10. D. Chaum, C. Crépeau and I. Damgård, “Multiparty Unconditionally Secure Protocols”, STOC ’98, pp. 11–19.
11. R. Canetti, U. Feige, O. Goldreich and M. Naor, “Adaptively Secure Multi-Party Computation”, STOC ’96, pp. 639–648.
12. J.H. Conway and N.J.A.Sloane, “Sphere Packings, Lattices and Groups”, Springer, 1993.
13. M.J. Coster, B.A. LaMacchia, A.M. Odlyzko, C.P. Schnorr and J. Stern, “Improved Low-Density Subset Sum Algorithms”, Journal of Computational Complexity, Vol. 2, 1992, pp. 111–128.
14. A. De Santis, Y. Desmedt, Y. Frankel and M. Yung, “How to Share a Function Securely”, STOC ’94, pp. 522–533.
15. Y. Desmedt and K. Kurosawa, “How to Break a Practical MIX and Design a New One,” EUROCRYPT ’00, pp. 557–772.
16. J. Feigenbaum and R. Ostrovsky, “A Note on One-Prover Instance-Hiding, Zero-Knowledge Proof Systems”, ASIACRYPT ’91, pp. 352–359.
17. M.R. Garey and D.S. Johnson, “Computers and Intractability: A Guide to the Theory of NP-Completeness,” W.H. Freeman and Company, 1979.
18. R. Gennaro, M. Rabin and T. Rabin, “Simplified VSS and Fast-track Multiparty Computations with Applications to Threshold Cryptography”, PODC ’98, pp. 101–111.

19. O. Goldreich, S. Micali and A. Wigderson, “How to Play Any Mental Game”, STOC ’87, pp. 218–229.
20. M. Jakobsson, “A Practical Mix”, EUROCRYPT ’98, pp. 448–461.
21. S. Kawamura and A. Shimbo, “Fast Server-Aided Secret Computation Protocols for Modular Exponentiation”, IEEE Journal on Selected Areas in Communications, Vol. 11, No. 5, 1993, pp. 778–784.
22. C.H. Lim and P.J. Lee, “Security and Performance of Server-Aided RSA Computation Protocols,” CRYPTO ’95, pp. 70–83.
23. C.H. Lim and P.J. Lee, “More Flexible Exponentiation With Precomputation”, CRYPTO ’94, pp. 95–107.
24. T. Matsumoto, H. Imai, C.S. Laih and S.M. Yen, “On Verifiable Implicit Asking Protocols for RSA Computation”, AUSCRYPT ’92, pp. 296–307.
25. T. Matsumoto, K. Kato and H. Imai, “Speeding Up Secret Computations with Insecure Auxiliary Devices”, CRYPTO ’88, pp. 497–506.
26. National Institute of Standards and Technology (NIST), “FIPS Publication 186-1: Digital Signature Standard”, December 15, 1998.
27. P. Nguyen and J. Stern, “The Hardness of the Hidden Subset Sum Problem and its Cryptographic Implications”, Crypto ’99, pp. 31–46.
28. B. Pfitzmann and M. Waidner, “Attacks on Protocols for Server-Aided RSA Computation”, EUROCRYPT ’92, pp. 153–162.
29. J.-J. Quisquater and M. De Soete, “Speeding up Smart Card RSA Computation with Insecure Coprocessors”, SMART CARD 2000, 1989, pp. 191–197.
30. E.M.Rains and N.J.A. Sloane, “Table of Constant Weight Binary Codes”, `http://www.research.att.com/~njas/codes/Andw/`, 2000.
31. T. Sander, A. Young and M. Yung, “Non-Interactive Crypto Computing for NC^1”, FOCS ’99, pp. 554–567.
32. C.-P. Schnorr, “Efficient Signature Generation for Smart Cards,” *Journal of Cryptology*, Vol. 4, 1991, pp. 161–174.
33. C.P. Schnorr and M. Euchner, “Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems”, Mathematical Programming 66, 1994, pp. 181–199.
34. C.P. Schnorr and H.H. Hörner, “Attacking the Chor-Rivest Cryptosystem by Improved Lattice Reduction”, EUROCRYPT ’95, pp. 1–12.
35. A.C. Yao, “Protocols for Secure Computations”, FOCS ’82, pp. 160–164.

A Privacy Analysis

A scheme is *private* if each set of the queries issued by it is indistinguishable from an equal-sized set of uniformly distributed random queries. We argue that if we choose a sufficient number of blinding factors in our scheme, then our scheme is private.

According to Section 5, an input vector $(k_1, ..., k_n)$ is transformed into the outsourced vector by applying *replication*, introducing *dependencies*, performing *blinding* and applying a permutation. For purposes of privacy analysis it suffices to consider only the permutation as well as the blinding process. The blindings are generated by selecting d of the blinding factors $r_1, ..., r_e$, i.e., $k'_i = k_i + \sum_{j=1}^{e} \gamma_{i,j} r_j$ with $\sum_{j=1}^{e} \gamma_{i,j} = d$ and $\gamma_{i,j} \in \{0, 1\}$. The permutation of the list k'_i is sent to the servers. Let us for sake of an argument assume that an attacker knows the random permutation. Then, the attacker can compute $g^{k'_i}/g^{k_i} = g^{\rho_i}$, where the exponent $\rho_i = \sum_{j=1}^{e} \gamma_{i,j} r_j$ $(1 \leq i \leq n)$ is unknown. Then, following the analyisis in [7], we know that:

Theorem 1. *Assuming that there exists an algorithm that, given g^{ρ_i}, for $\rho_i = \sum_{j=1}^{e} \gamma_{i,j} r_j$, and $1 \leq i \leq n-1$, computes the discrete log of g^{ρ_n} with success rate ϵ, then there is an algorithm to compute the discrete log on arbitrary inputs in expected time $O(1/\epsilon)$ with success rate ϵ.*

However, and just as in [7], there is a further complication in that one component of the signature is a linear function of the secret value k, and the secret key x. This opens up for a host of problems, which we will list and elaborate on in the following. We note, though, that there is no known proof of security, as there are no proofs of the bounds on the complexity of the attacks to be described, and others that are not yet discovered. A reasonable heuristic, though, can be obtained from studying the progress of these attacks.

Attack Type I: Lattice Based Attacks. Since we reuse blinding elements for the different blinding factors, we need to consider the impact of lattice based attacks on our security. In particular, one of these attacks relates to the classical subset sum problem:

Subset Sum Problem: *For given positive integers $w_1, \ldots, w_N$ (the weights) and the sum S, find a subset of the weights, i.e., determine the variables $x_1, \ldots x_N \in \{0, 1\}$ such that $S = \sum_{i=1}^{N} x_i w_i$.*

Transforming the problem into a lattice problem, i.e., reducing it to the problem of finding a shortest vector in a lattice [13] is the most powerful method known to date to attack the classical subset sum problem. In practice, lattice basis reduction methods such as the LLL reduction algorithm or the Blockwise-Korkine-Zolotarev (BKZ) reduction algorithm [13,33,34] provide suitable approximations to the shortest lattice vector. They perform well (in respect to run-time and quality of the approximation) for a small number of weights but break down for $N > 200$. While the LLL algorithm has a binary complexity of $O(N^6 \log{}^3 B)$ (where B is a bound on the size of the entries of the corresponding lattice basis) there is no polynomial time algorithm known to date for performing a BKZ reduction. On the other hand, BKZ reduction yields better reduction results than the LLL algorithm. For an increasing number of weights the quality of the approximation becomes insufficient for providing a solution to the subset sum problem.

Moreover, since according to the construction of the blinding not only the variables $x_1, \ldots x_N \in \{0, 1\}$ but also the weights $w_1, \ldots, w_N$ are unknown, the attacker is faced with a problem that is even harder to solve:

Hidden Sum Problem: *For a given N and sums $S_1, \ldots, S_l \in \mathbb{Z}_p$, find weights $w_1, \ldots, w_N \in \mathbb{Z}_p$ and determine the variables $x_{1,1}, \ldots x_{N,l} \in \{0, 1\}$ such that $S_j = \sum_{i=1}^{N} x_{i,j} w_i$ for $1 \leq j \leq l$.*

This problem was first introduced in [7]. Apart from exhaustive search, lattice-based methods are once again the most efficient techniques known to date to solve the problem [27]. However, regardless of the density of the subset sum problem neither of these methods is practical for attacking the privacy of our scheme of server-aided signature generation. This is due to efficiency reasons. Both exhaustive search and lattice-based attacks can be performed only for small problem instances with a very few weights. E.g., in [27] the problem could only be solved for up to $N \approx 45$ whereas we consider settings with $N > 80000$.

Attack Type II: Cancellations among Blindings. If an attacker could efficiently select a small number of signatures such that the applied blindings cancel each other, he would be also be able to retrieve the secret key used.

1. *Two identical sets of blinding elements.*
   This case, namely that in which $\{\rho_{1,(\mathcal{G}_3)_i}, \ldots \rho_{4,(\mathcal{G}_3)_i}\} = \{\rho_{1,(\mathcal{G}_3)_j}, \ldots \rho_{4,(\mathcal{G}_3)_j}\}$ *for* $i \neq j$ is efficiently prevented by the use of enumeration for the selection, which guarantees that no two queries will use the very same set of blinding elements. Thus, for a given input size n, the number of blinding factors has to be chosen such that $\binom{e}{d} \geq n$.
2. *The sums of two blinding sets are equal.*
   This case, namely $\rho_{1,(\mathcal{G}_3)_i} + \ldots + \rho_{4,(\mathcal{G}_3)_i} \bmod q = \rho_{1,(\mathcal{G}_3)_j} + \ldots + \rho_{4,(\mathcal{G}_3)_j} \bmod q$ can be seen to occur only with probability $\frac{1}{q}$, which is negligible.
3. *All blinding elements are selected.*
   We note that it is not possible to cancel the blinding factors by adding all of the resulting queries. This is so even if all the possible enumerations were to be used, since the elements of the linear combination with necessity must have different signs in order for all the blinding elements to cancel.
4. *The attacker identifies more than one but less than all elements.*
   We consider in the following a selection of blinding elements such that any two selections have a Hamming distance of at least 2. In the following, let $B_1, B_2, B_3, \ldots$ be the sets with four blinding factors $b_{i,1}, b_{i,2}, b_{i,3}, b_{i,4}$ $(i = 1, 2, 3, \ldots)$. Apparently, an attacker won't succeed by only choosing two sets of blinding factors B_1 and B_2 since the two sets differ in at least one element, and therefore cannot cancel. Thus, in each linear combination of those two sets

$$\begin{aligned} a_1 \cdot (b_{1,1} + b_{1,2} + b_{1,3} + b_{1,4}) + \\ a_2 \cdot (b_{2,1} + b_{2,2} + b_{2,3} + b_{2,4}) \bmod q \end{aligned}$$

   with $a_1, a_2 \in \mathbb{Z}_q^*$, there will be at least a sum of two elements left. In case of three sets, an attacker is faced with the problem of finding a third set to cancel the sum of the first two. Since the linear combination of the first two will either contain elements with different signs or with different factors, choosing only three sets is not sufficient either. Thus, an attacker will have to choose at least four suitable sets.

There are four cases to be considered; one in which the four chosen sets elements correspond to four different signatures; one in which they correspond to three; then two, with two elements for each; then two, with one element for one and three elements for the other. We will focus on the probability of success for the first of these cases. The probability of success for the others will be smaller, as can be seen from case analysis.

For the first case, the probability of success corresponds to an attack in which the adversary selects three elements (of the outsourced type) at random from all available such elements, and then selects the last one, hoping that the four elements he chose are such that the blinding factors cancel. Then, he selects four signatures to match these four outsourced elements. The probability of the fourth outsourced element having the desired property is $1/(\varDelta - 3)$, where $\varDelta = 3(n+1) < \binom{e}{4}$. The probability of selecting the matching signatures is not better than $1/\binom{n}{4}$. Thus, the success probability of the adversary is smaller than $8/(n-3)^5$. We see that for $n > 99337$ the attacker has a chance less than 2^{-80} to successfully retrieve the private key.

It is left to be shown that for any given three sets of blinding factors B_1, B_2 and B_3, there is at most one fourth set B_4 such that all four are linear dependent, i.e.,

$$\sum_{i=1}^{4} a_i \cdot (b_{i,1} + b_{i,2} + b_{i,3} + b_{i,4}) \bmod q = 0$$

with $a_1, a_2, a_3, a_4 \in \mathbb{Z}_q^*$. Linear dependence of B_1, B_2, B_3 and B_4 can only be achieved if not more than four elements in the linear combination of the given B_1, B_2 and B_3 are left and each element is weighted with the same factor. In the following let the first set of blinding elements be multiplied by factor a_1, the second one with a_2 and the third one with a_3.

(a) If each set contains at least one element that is not contained in any other set, then $a_1 = a_2 = a_3$, since the results in the linear combination will have to have the same factor. But since the sets differ by at least one element, the resulting combination will contain at least 5 elements, thus the cancellation cannot be achieved with only a fourth vector.

(b) In case that only two sets contain elements that are not found in the other sets (where those sets in the linear combination are weighted with a_1 and a_2 respectively), then $a_1 = a_2$. Moreover, there will be at least one element in the sum that is weighted with $a_1 + a_3$ and one with $a_1 + a_2$. Since $a_2 \neq 0$ and $a_3 \neq 0$, the chances that a fourth vector can be found are only in case $a_3 = -a_2 = -a_1$. Thus, the combination of these three vectors consists of only zeros or a_1's. If the number of a_1's is four, there is a fourth vector to achieve cancellation, otherwise there isn't.

(c) In the last case, that only one set (weighted with a_1 in the linear combination) contains elements that are not contained in any of the two other sets, the elements in the linear combination will need to match to zero or a_1. Once again, there is at least one factor $a_1 + a_3$ and at lest one $a_1 + a_2$. With $a_2 \neq 0$ and $a_3 \neq 0$ it follows that in order to keep the chance that a fourth vector can be found, $a_2 = a_3 = -a_1$. If there is a factor $a_1 + a_2 + a_3$ or more than 4 elements left, the fourth vector cannot be found.

This completes the argument. For the smartcard setting, a corresponding argument can be applied for $\Delta = (8+1)(n'+1)$, where n' is the cumulative batch size.

B Robustness Analysis

A scheme is *robust* if it is not possible for an adversary controlling all the computing servers to produce replies that cause an incorrect output to be generated by the originator. In the following let the vector $\mathcal{G}_3$ (see Section 5) be arranged in a $(n+1) \times 3$ dimensional matrix, where the elements of the first and second row are the $k_1, \ldots k_{n+1}$ and the third row contains the corresponding checksums. By case analysis over the number of changed columns, we argue that in order to succeed with an attack, it is necessary for an attacker to select a set of computational queries of size at least six from the total $3n+3$ issued queries, assuming $n \geq 11$. In the following the offsets are described in terms of additions since the results of applying any arbitrarily chosen function can also be achieved by means of performing an addition in Z_q.

1. *One or two altered columns.* Let i be the first column (corresponding to the pre-permutation stage) in which incorrect values are returned. Assume that the non-checksum values and their duplicates are offset by an integer δ. Then, by the recurrence describing how checksums are computed, the checksum in column i needs to be offset by δ, or the resulting checksum verification will fail. Then, turning to the values of column $i+1$, we see that the difference of the checksum of this column and one of its two non-checksum values has to be offset by -2δ in order for the verification of the checksum of this column to succeed. i.e., If we alter all the values of column $i+1$, that gives us a total of six values to select. If only the non-checksum value and its duplicate are altered, the sum of the checksum and the non-checksum value of the $i+2$nd column has to be offset by 2δ, thus at least six values have to be selected. If only the checksum value of the $i+1$st column is altered, i.e., offset by -2δ, the sum of the non-checksum and checksum values of column $i+2$ will then in turn have to be offset by 2δ in turn. (By the same argument as above, we would then have to correct the $i+2$nd column in the same manner as the $i+1$st.)
2. *More than two, less than n incorrect columns.* Since each original value is duplicated in our scheme, an attacker needs to alter at the very least $2k$ queries in order to corrupt the computation of k values, or the results and their duplicates will not be consistent. (This is so since the original values are not dependent.) For "small" values of k, this amounts to at least six elements, given the restriction $k > 2$. For "large" values of k, the adversary alters all but a few queries, in which case he has to select those which he

will not alter. In order not to affect $n-k$ original values, he needs to select the queries of these values, and the corresponding checksums. Since there remains $3k+3$ values (where the last portion is due to the added $n+1$st column) this gives us that at least six values have to be selected, given $k>0$.

3. *All values incorrect.* Let us consider a few cases: (1) All non-checksum values are *replaced* by the same value. (2) All non-checksum values are *offset* by the same value. (3) All values to be computed (not necessarily including the checksums) are altered.

   In the first case, we see from the recurrence equation describing how the checksums are computed (and the fact that all non-checksum values are set to the same value) that the absolute value of all the checksum values will equal the first checksum value, but with alternating signs for each column. Therefore, unless the the non-checksum values are set to zero (which would cause all replies to equal one, and which would therefore not be accepted), it is necessary for the adversary to partition the set of values into at least two sets. As long as $n \geq 11$, this means that the adversary has to select at least select six values from all the values to be computed.

   In the second case, by the argument above, all checksums have to be offset by either zero or non-zero. The former would cause no change, and can therefore be ignored; the latter would force the adversary to correctly partition the set, giving us the same result as above.

   If all values to be computed are altered then either we need to alter then in the same manner, or we need to partition them into two or more sets, each of which will be altered in a particular manner. Consider the smallest set altered in a particular manner. If it contains six or more elements, we are done. Due to the use of duplication, it cannot contain an odd number. Assume it contains four elements. Given the way the checksums are computed, we know from above that either the two checksums in these two columns have to be altered then, or at least one element in each of the adjacent columns. This again gives us that at least six elements that have to be modified. We are left with the case in which the smallest set contains two elements. If there are two such sets, this gives the same result as if the smallest set were of size four. If there is only one set, it is an identical setting to the argument of "one or two added columns". This concludes the argument.

We argued in Appendix A that each set of query values has a distribution that cannot be distinguished from a uniform at random distribution by the attacker. Therefore, and since a total of $3n+3$ computational queries will be issued for an input size corresponding to n signature generations, the probability of success for the adversary will be bounded by $1/(3\cdot\binom{3n+2}{5})$. To obtain a sufficient robustness for the smartcard setting with batch size $n=20$, the original vector has to be replicated 7 times (making its size eight times that before replication), thus the attacker will have to choose $(8+1)\cdot 2$ elements.

C Efficiency Analysis

The global cost, i.e., the amount of computations to be performed by the external servers, is determined by the size of the delegated vector and amounts to $3n + 3$ exponentiations.

Turning now to the amount of local computation, we can see that for each exponentiation the originator wants to have performed, he will have to perform the following number of multiplications:

$$\begin{array}{lcl} & 2 & \text{add'l mult. due to introduced dependencies} \\ + & 3 \cdot 2 + \frac{1}{n}(200 \cdot 75 + \frac{75(75-1)}{2})) & \text{add'l mult. due to blinding} \\ \hline \approx & 8 & \text{add'l multiplications} \end{array}$$

In comparison, addition chain methods for exponentiations require about 10 multiplications per signature for batches of size approximately 100000 signatures [5].

In the smartcard setting, the exponentiation work for generating the blinding factors is performed in a pre-computation step (but the pairing of the blinding factors is still not.) This results in local costs per signature of approximately $2 + 9 \cdot 4 = 38$ multiplications. Since window based methods require about 200 multiplications per signature for batches of size 20, this corresponds to a speed-up of 81%.