DISTRIBUTION OF MODULAR SUMS AND THE SECURITY OF THE SERVER AIDED EXPONENTIATION

Phong Q. Nguyen
Département d'Informatique, École Normale Supérieure
45, rue d'Ulm, 75230 Paris Cedex 05, France
pnguyen@ens.fr

Igor E. Shparlinski
Department of Computing, Macquarie University
Sydney, NSW 2109, Australia
igor@comp.mq.edu.au

Jacques Stern
Département d'Informatique, École Normale Supérieure
45, rue d'Ulm, 75230 Paris Cedex 05, France
stern@dmi.ens.fr

Abstract

We obtain some uniformity of distribution results for the values of modular sums of the form

$$\sum_{j=1}^{n} a_j x_j \pmod M \qquad (x_1, \ldots, x_n) \in \mathcal{B}$$

where $M \geq 1$ is an integer, $a_1, \ldots, a_n$ are elements of the residue ring modulo M, selected unformly at random, and $\mathcal{B}$ is an arbitrary set of n-dimensional integer vectors. In some partial cases, for very special sets $\mathcal{B}$, some results of this kind have been known, however our estimates are more precise and more general. Our technique is based on fairly simple properties of exponential sums. We also give cryptographic applications of some of these results. In particular, we consider an extension of a pseudo-random number generator due to V. Boyko, M. Peinado and R. Venkatesan, and establish the security of some discrete logarithm based signature schemes making use of this generator (in both its original and extended forms). One of these schemes, which uses precomputation is well known. The other scheme which uses server aided computation, seems to be new. We show that for a certain choice of parameters one can guarantee an essential speed-up of both of these schemes without compromising the security (compared to the traditional discrete logarithm based signature scheme).

1 Introduction

Let $M \geq 2$ be integers. We denote by $\mathbb{Z}_M$ the residue ring modulo M.

For a given n-dimensional vector $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_M$ and a given set of n-dimensional vectors $\mathcal{B} \subseteq \mathbb{Z}_M^n$ we consider the distribution of *modular sums*

$$\mathbf{a} \cdot \mathbf{x} \equiv \sum_{j=1}^{n} a_j x_j \pmod M, \qquad (x_1, \ldots, x_n) \in \mathcal{B}.$$

More precisely, given a $c \in \mathbb{Z}_M$ and an integer $h \geq 1$, we denote by $N_{\mathbf{a}}(\mathcal{B}, c)$ the number of solutions of the congruence

$$\mathbf{a} \cdot \mathbf{x} \equiv c \pmod M, \qquad \mathbf{x} \in \mathcal{B}.$$

Because for many applications it is technically more useful to work with probabilities we define

$$P_{\mathbf{a}}(\mathcal{B}, c) = \frac{1}{|\mathcal{B}|} N_{\mathbf{a}}(\mathcal{B}, c)$$

as the probability that $\mathbf{a} \cdot \mathbf{x} \equiv c \pmod M$ for a random vector $\mathbf{x}$ chosen uniformly from $\mathcal{B}$.

We use exponential sums to show that for almost all vectors $\mathbf{a} \in \mathbb{Z}_M^n$ these quantities take their expected values under some natural restrictions on the cardinality of $\mathcal{B}$.

Although this holds to arbitrary sets $\mathcal{B}$, for our applications we are mainly interested in the following special sets. Given an integer $h \le M - 1$ we define $\mathcal{B}_{n,h}$ as the set of integer vectors $\mathbf{x} = (x_1, \ldots, x_n)$ with $0 \le x_j \le h-1$, $j = 1, \ldots, n$. Given an integer k, $n \ge k \ge 1$, we also define $\mathcal{B}_{n,k,h}$ as the subset of $\mathcal{B}_{n,h}$ consisting of vectors with precisely k non-zero components. Hence

$$|\mathcal{B}_{n,h}| = h^n \qquad \text{and} \qquad |\mathcal{B}_{n,k,h}| = (h-1)^k \binom{n}{k}.$$

In the important (although not always optimal for our applications) special case $h = 2$ we put

$$\mathcal{B}_{n,2} = \mathcal{B}_n \qquad \text{and} \qquad \mathcal{B}_{n,k,2} = \mathcal{B}_{n,k}.$$

In the case $\mathcal{B} = \mathcal{B}_n$ several results of this kind have been known, see [1, 8, 9, 12, 14]. These results have found several cryptographic applications, in particular, to some cryptosystems proposed in [2], see [14]. However for other sets no similar results seemed to be known. In particular for sets $\mathcal{B}_{n,k}$ this problem has been mentioned in [14].

We also give some cryptographic motivations and applications of our results, in particular in the signature scheme with precomputation [2].

In many discrete logarithm based protocols, one needs to generate pairs of the form $(x, g^x \pmod p)$ where x is random and g is a fixed base. The El Gamal [5] and DSA [13] (*Digital Signature Algorithm*) signatures as well as the Schnorr [18, 19] and Brickell–McCurley [4] identification and signature schemes are examples of such protocols. The generation of these pairs is often the most expensive operation, which makes it tempting to reduce the number of modular multiplications required per generation, especially for smartcards. There are basically two ways to solve this problem. One way is to generate separately a random x, and then to compute $g^x \pmod p$ using a precomputation method [3, 7, 16, 10].

The other way is to generate x and $g^x \pmod p$ together by a special

pseudo-random number generator which also uses precomputation. Schnorr [18] was the first to propose such a preprocessing scheme. The scheme has much better performances than all other methods but there is a certain drawback: the output exponent x is no more guaranteed to be random, and therefore, each generation might leak information. Indeed, de Rooij [15] showed how to break the scheme. A later modification proposed by Schnorr in [19], was also broken by de Rooij [17].

Boyko, Peinado and Venkatesan [2] have recently proposed a new and very simple generator to produce pairs of the form $(x, g^x \pmod p)$, where p is a prime number, and $g \in \mathbb{Z}_p^*$ is of multiplicative order M.

We describe a generalization of the **BPV** generator, which combined with a certain exponentiation algorithm from [3], becomes computationally more efficient.

This generator as well as our extension, can naturally be used for a signature scheme with precomputation, and thus provides a substantial speed-up compared to the traditional scheme. However, it should be mentioned that the approach of [10] seems to be more efficient for the above values of M (although the asymptotic behaviour of that method has not been evaluated in that paper).

On the other hand, we also describe another scheme, to which the results of [10] do not apply. In that scheme, instead of precomputing and thus storing a rather substantial set of integers, one uses the server to assist the signature generation. This new scheme is suitable for computation on a device with low computational power and memory (such as a smart-card, for example).

We show that the security of both signature schemes is preserved, when certain conditions on their parameters are met. Our proof is based on the notion of statistical distance.

Throughout the paper $\log z$ and $\ln z$ denote the binary and the natural logarithm of real $a > 0$, respectively.

2 Extended BPV Generator

Let $g \in \mathbb{Z}_p^*$ be of multiplicative order M.

The original generator of Boyko, Peinado and Venkatesan [2], which we call **BPV**, with integer parameters $n \geq k \geq 1$ can be described as follows:

Preprocessing Step: Generate n random integers $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}_M$. For each $j = 1, \ldots, n$, compute $\beta_j \equiv g_j^{\alpha} \pmod p$ and store the values of α_j and β_j in a table.

Pair Generation: Whenever a pair (x, g^x) is needed, randomly generate $S \subseteq \{1, \ldots, n\}$ such that $|S| = k$. Compute

$$x \equiv \sum_{j \in S} \alpha_j \pmod M \qquad \text{and} \qquad X \equiv \prod_{j \in S} \beta_j \equiv g^b \pmod p.$$

If $x \equiv 0 \pmod M$ then start again, otherwise return the pair (x, X).

For any output (x, X), we indeed have $X \equiv g^x \pmod p$. The scheme needs to store n elements of $\mathbb{Z}_M$, and n elements of $\mathbb{Z}_p^*$. It requires $k - 1$ modular multiplications to compute X (and the same number of additions to compute x, but this cost is negligible). This can be compared with the cost of direct computation of g^x which is about $1.5 \log M$ modular multiplications on average and about about $1.5 \log M$ modular multiplications in the worst case. Thus the ratio $k/\log M$ is a natural measure of speed-up of the **BPV** generator.

Recall that for the DSA [13] and Schnorr [18, 19] schemes M has 160 bits, while for the El Gamal [5] and Brickell–McCurley [4] schemes M has at least 512 bits. Each generation requires k modular multiplications. For $M = p - 1$ where p is a 512-bit prime the authors of [2] suggest to take $n = 512$ and $k = 64$.

We now describe an extension of the **BPV** generator which we will call **EBPV**, with the same integer parameters $n \geq k \geq 1$ and another integer parameter $h \geq 1$. The preprocessing step of the generator is the same however the pair generation follows a slightly more general algorithm:

Extended Pair Generation: Whenever a pair (x, g^x) is needed, randomly generate $S \subseteq \{1, \ldots, n\}$ such that $|S| = k$ and for each $j \in S$ select a random integer $x_j \in \{1, \ldots, h-1\}$. Compute

$$x \equiv \sum_{j \in S} \alpha_j x_j \pmod{M} \qquad \text{and} \qquad X \equiv \prod_{j \in S} \beta_j^{x_j} \pmod{p},$$

thus $X \equiv g^x \pmod{p}$. If $x \equiv 0 \pmod{M}$ then start again, otherwise return the pair (x, X).

It is easy to see that we still have $X \equiv g^x \pmod{p}$ and that for $h = 2$ this is precisely the **BPV** generator.

The cost of computing x is still very low at least if h is small (it involves $k-1$ modular additions and $k-1$ modular multiplications but one of the factors is small). It has been shown in Theorem 1 of [3] (see also Theorem 7 of [7]) that there exists a (simple) algorithm which computes X with only $k + h - 3$ modular multiplications.

We finally explain how to use the above technique in a completely different scenario : instead of precomputing and thus storing a rather substantial set of integers, one uses the server to help with random pair generation. Applications that we have in mind are related with devices whose computational power and memory are extremely low (such as a smart-card, for example). Such devices can optionnally be plugged in a base so as to communicate with a server (a PC which may or may not be connected to the Internet). We use the server in the following way:

Loading Step - server side: The server generates n random integers $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}_M$. For each $j = 1, \ldots, n$, it computes $\beta_j \equiv g_j^{\alpha} \pmod{p}$ and broadcasts the values of α_j and β_j.

Loading Step - client side: The server randomly generates $S \subseteq \{1, \ldots, n\}$ such that $|S| = k$ and, while receiving the server's communication, the client stores the k pairs (α_j, β_j) corresponding to the indices j in S.

After this, one applies the above *extended pair generation* procedure and stores the pair (x, X) for further signature generation.

It should be noted that, in the special case where $h = 2$, the two steps that have to be performed by the server can be merged since the required multiplications can be performed on the fly. This definitely reduces the overall memory needed.

We believe that our scheme is realistic and, in Section 6, we will propose suitable choices for the parameters that should allow practical implementations.

However, some precautions are in order. Firstly, it should be impossible to detect the choices of indices performed by the client to form S. Otherwise, the pair (x, X) used at signature generation becomes known to the attacker and, as is well known, this allows to disclose the secret signing key. Secondly, the server should be trusted. Otherwise, there are easy strategies that disclose the signing key again: for example, one can feed the device with n identical pairs. On the other hand, we do not require the various pairs (α_j, β_j) to remain secret since we only care about the randomness of the output pair.

Smart cards manufacturers have developped countermeasures against "timing" and "power" attacks, which should hopefully address our first concern. As for the second, it can be handled by various cryptographic means. If there is a communication link from the card to the server, both can perform a key exchange and the server can encipher the communication. There are ways to achieve this and keep the computational overhead low for the server. In a "broadcast" scenario, this becomes impossible and the only way seems to have the server hash-and-sign the broadcast. Again, this can be done by keeping the computing power of the client low, for example, by using low-exponent RSA. Still, it might be the case that the device cannot hash on the fly and keep up with the communication speed. A way to overcome the difficulty is to use hash-trees: once the pairs have been sent, the full hash-tree built from the n pairs is broadcast, together with the signature of its root. The client only needs to capture the k paths requested for checking the hash computations and the signature.

3 Distribution of Modular Sums

Let us define

$$\mathbf{e}(z) = \exp(2\pi i z/M).$$

We make use of the identity

$$\sum_{\lambda=0}^{M-1} \mathbf{e}(\lambda u) = \begin{cases} 0, & \text{if } u \not\equiv 0 \pmod M; \\ M, & \text{if } u \equiv 0 \pmod M; \end{cases} \tag{1}$$

which holds for any integer u and which follows from the formula for the sum of a geometric progression (see Exercise 11.a Chapter 3 of [20]).

Lemma 1 *For any integer* $\lambda \not\equiv 0 \pmod M$*,*

$$\sum_{\mathbf{a}\in\mathbb{Z}_M^n} \left| \sum_{\mathbf{x}\in\mathcal{B}} \mathbf{e}\left(\lambda\,\mathbf{a}\cdot\mathbf{x}\right) \right|^2 = M^n|\mathcal{B}|.$$

Proof. We have

$$\sum_{\mathbf{a}\in\mathbb{Z}_M^n} \left| \sum_{\mathbf{x}\in\mathcal{B}} \mathbf{e}\left(\lambda\,\mathbf{a}\cdot\mathbf{x}\right) \right|^2 = \sum_{\mathbf{x},\mathbf{y}\in\mathcal{B}} \sum_{\mathbf{a}\in\mathbb{Z}_M^n} \mathbf{e}\left(\lambda\left(\mathbf{a}\cdot\mathbf{x} - \mathbf{a}\cdot\mathbf{y}\right)\right).$$

If $\mathbf{x} = \mathbf{y}$ the contribution of the inner sum is M^n. If $\mathbf{x} \neq \mathbf{y}$ then, assuming without loss of generality that $x_n \neq y_n$, we obtain

$$\sum_{\mathbf{a}\in\mathbb{Z}_M^n} \mathbf{e}\left(\lambda\left(\mathbf{a}\cdot\mathbf{x} - \mathbf{a}\cdot\mathbf{y}\right)\right)$$

$$= \sum_{(a_1,\ldots,a_{n-1})\in\mathbb{Z}_M^{n-1}} \mathbf{e}\left(\lambda\sum_{j=1}^{n-1} a_j(x_j - y_j)\right) \sum_{a_n=0}^{M-1} \mathbf{e}(\lambda a_n(x_n - y_n)) = 0$$

and the desired result follows. □

Using the Cauchy inequality, we derive from Lemma 1 that

$$\sum_{\mathbf{a}\in\mathbb{Z}_M^n} \left| \sum_{\mathbf{x}\in\mathcal{B}} \mathbf{e}\left(\lambda\,\mathbf{a}\cdot\mathbf{x}\right) \right| \leq M^n|\mathcal{B}|^{1/2}. \tag{2}$$

Theorem 2 *The identity*

$$\frac{1}{M^n} \sum_{\mathbf{a}\in\mathbb{Z}_M^n} \sum_{c\in\mathbb{Z}_M} \left(P_{\mathbf{a}}(\mathcal{B},c) - \frac{1}{M}\right)^2 = \frac{M-1}{M|\mathcal{B}|}$$

holds.

Proof. From (1) we have

$$N_{\mathbf{a}}(\mathcal{B},c) = \frac{1}{M} \sum_{\mathbf{x}\in\mathcal{B}} \sum_{\lambda=0}^{M-1} \mathbf{e}\left(\lambda\left(\mathbf{a}\cdot\mathbf{x} - c\right)\right).$$

The contribution of the term corresponding to $\lambda = 0$ is $|\mathcal{B}|/M$.

Therefore

$$
\begin{aligned}
\sum_{c\in\mathbb{Z}_M}\left(N_{\mathbf{a}}(\mathcal{B},c)-\frac{|\mathcal{B}|}{M}\right)^2 &= \sum_{c\in\mathbb{Z}_M}\left(\frac{1}{M}\sum_{\lambda=1}^{M-1}\mathbf{e}\left(-\lambda c\right)\sum_{\mathbf{x}\in\mathcal{B}}\mathbf{e}\left(\lambda\,\mathbf{a}\cdot\mathbf{x}\right)\right)^2\\
&= \frac{1}{M^2}\sum_{c\in\mathbb{Z}_M}\sum_{\lambda,\eta=1}^{M-1}\mathbf{e}\left(-c(\lambda+\eta)\right)\sum_{\mathbf{x},\mathbf{y}\in\mathcal{B}}\mathbf{e}\left(\lambda\,\mathbf{a}\cdot\mathbf{x}+\eta\,\mathbf{a}\cdot\mathbf{y}\right)\\
&= \frac{1}{M^2}\sum_{\lambda,\eta=1}^{M-1}\sum_{\mathbf{x},\mathbf{y}\in\mathcal{B}}\mathbf{e}\left(\lambda\,\mathbf{a}\cdot\mathbf{x}+\eta\,\mathbf{a}\cdot\mathbf{y}\right)\sum_{c\in\mathbb{Z}_M}\mathbf{e}\left(-\lambda c-\eta c\right).
\end{aligned}
$$

The sum over c vanishes if $\lambda+\eta\not\equiv 0 \pmod M$ and is equal to M otherwise. Hence,

$$
\begin{aligned}
\sum_{c\in\mathbb{Z}_M}\left(N_{\mathbf{a}}(\mathcal{B},c)-\frac{|\mathcal{B}|}{M}\right)^2 &= \frac{1}{M}\sum_{\lambda=1}^{M-1}\sum_{\mathbf{x},\mathbf{y}\in\mathcal{B}}\mathbf{e}\left(\lambda\left(\mathbf{a}\cdot\mathbf{x}-\ \mathbf{a}\cdot\mathbf{y}\right)\right)\\
&= \frac{1}{M}\sum_{\lambda=1}^{M-1}\left|\sum_{\mathbf{x}\in\mathcal{B}}\mathbf{e}\left(\lambda\,\mathbf{a}\cdot\mathbf{x}\right)\right|^2.
\end{aligned}
$$

Applying Lemma 1, we obtain the desired result. □

In particular, using the Cauchy inequality we obtain

$$
\frac{1}{M^n}\sum_{\mathbf{a}\in\mathbb{Z}_M^n}\sum_{c\in\mathbb{Z}_M}\left|P_{\mathbf{a}}(\mathcal{B}_{n,k,h},c)-\frac{1}{M}\right|\le M^{1/2}|\mathcal{B}_{n,k,h}|^{-1/2}. \tag{3}
$$

4 Statistical distance

In complexity theory, it is customary to use the notion of statistical distance between two distributions (see for instance [11, 6]). Recall that the (probability) distribution $\mathcal{D}$ associated with a random variable X over a set S is the function mapping any n of S to $\mathcal{D}(n)=\Pr(X=n)$. The *statistical distance* between two distributions $\mathcal{U}$ and $\mathcal{V}$ over a finite set S is defined as:

$$
L(\mathcal{U},\mathcal{V})=\frac{1}{2}\sum_{n\in S}\left|\mathcal{U}(n)-\mathcal{V}(n)\right|.
$$

It is indeed a distance.

We will need to consider sampled distributions.

Denote by $\mathcal{D}^m$ the distribution defined over S^m by choosing independently at random m samples of S, according to $\mathcal{D}$. In other words, $\mathcal{D}^m(n_1, \ldots, n_m) = \mathcal{D}(n_1)\mathcal{D}(n_2)\cdots\mathcal{D}(n_m)$. We will use the following elementary result, which shows that, when two distributions are close, the sampled distributions remain close:

Lemma 3 *Let $\mathcal{U}$ and $\mathcal{V}$ be two distributions defined over a set S. Then for all integer m:*

$$L(\mathcal{U}^m, \mathcal{V}^m) \leq mL(\mathcal{U}, \mathcal{V}).$$

Proof. Define $m+1$ hybrid distributions $\mathcal{D}_k$ over S^m by choosing independently k elements of S according to $\mathcal{U}$, and $m-k$ according to $\mathcal{V}$. That is, we let for k in $\{0, \ldots, m\}$:

$$\mathcal{D}_k(n_1, \ldots, n_m) = \prod_{1 \leq i \leq k} \mathcal{U}(n_i) \times \prod_{k < i \leq m} \mathcal{V}(n_i).$$

In particular, the distributions $\mathcal{D}_0$ and $\mathcal{D}_m$ are respectively identical to $\mathcal{U}^m$ and $\mathcal{V}^m$. From the triangle inequality, we have:

$$L(\mathcal{U}^m, \mathcal{V}^m) \leq \sum_{k=0}^{m-1} L(\mathcal{D}_k, \mathcal{D}_{k+1}).$$

We conclude by noting that each distance $L(\mathcal{D}_k, \mathcal{D}_{k+1})$ is actually equal to $L(\mathcal{U}, \mathcal{V})$. □

5 Security of Signature Schemes with the EBPV Generator

Any signature scheme consists of three algorithms: a key generation algorithm $\mathcal{A}_{\text{key}}$, a signature algorithm $\mathcal{A}_{\text{sig}}$, and a verification algorithm $\mathcal{A}_{\text{ver}}$. When a preprocessing generator is used, the key generation and signature verification algorithms remain the same, but one adds a preprocessing generation algorithm $\mathcal{A}_{\text{pre}}$.

Let $\mathcal{A}^*_{\text{sig}}$ be the new signature algorithm using the generator, which takes as input the signature keys, the message to sign, and the precomputations generated by $\mathcal{A}_{\text{pre}}$.

We restrict the definitions to the cases we are interested in (such as the El Gamal, DSA, Schnorr and similar signature schemes), by clarifying the link between $\mathcal{A}_{\text{sig}}$ and $\mathcal{A}^*_{\text{sig}}$. The security proof is only valid for that model, which we now make precise. At each signature request, the algorithm $\mathcal{A}_{\text{sig}}$ calls a probabilistic oracle outputting a value of the form $(x, f(x))$ where x is uniformly distributed over the set of keys and f is a function given on this set. The value is then used to produce a signature. For instance, with the DSA [13] and Schnorr [19] signature schemes one has $f(x) \equiv g^x \pmod p$ and the set of keys is $\mathbb{Z}^*_M$, where M is a 160-bit prime divisor of $p-1$. The bit-size of p varies from 512 to 1024.

The algorithm $\mathcal{A}^*_{\text{sig}}$ differs from $\mathcal{A}_{\text{sig}}$ only by the oracle called. The new probabilistic oracle, that is the generator, always outputs a value of the form $(x, f(x))$ with the same function f, but this time, the distribution of x is not necessarily uniform. We will call *defect* Δ of the generator the statistical distance between its distribution and the uniform distribution.

The oracle of $\mathcal{A}_{\text{sig}}$ only takes the keys as input, whereas the generator of $\mathcal{A}^*_{\text{sig}}$ takes as input both the keys and the precomputations.

Note that the defect of the **EBPV** generator is equal to

$$\Delta_{\mathbf{a}}(n,k,h) = \frac{1}{2} \sum_{c \in \mathbb{Z}_M} \left| P_{\mathbf{a}}(\mathcal{B}_{n,k,h}, c) - \frac{1}{M} \right| \tag{4}$$

and thus can be estimated using (3).

We now model an attacker, with respect to the most powerful attacks, namely adaptive attacks (see [6]). We call an *adaptive attack* with m chosen messages any algorithm which makes exactly m requests to the signature algorithm on messages of his choice, during an execution. Such an attack is said to succeed an *existential forgery* if it produces a valid signature of a message of which it had not requested a signature. For any adaptive attack $\mathcal{T}^*$ against the speeded-up signature scheme, one can associate an adaptive attack $\mathcal{T}$ against the original signature scheme, by replacing each call to $\mathcal{A}^*_{\text{sig}}$ by a call to $\mathcal{A}_{\text{sig}}$, with the same parameters. The attacks therefore have the same complexity (remember that each call to the signature algorithm contributes only by one unit to the overall complexity of the attack).

Theorem 4 *Let $\mathcal{T}^*$ be an adaptive attack with m chosen messages against the signature scheme using the* **EBPV** *generator with parameters n, k and*

h. If $\mathcal{T}$ is the adaptive attack corresponding to the original signature scheme, then the probabilities $P_{\mathbf{a}}(n,k,h,m,\mathcal{T}^*)$ and $P(m,\mathcal{T})$ of success of existential forgery for identical choices of the keys for both attacks
satisfy

$$\frac{1}{M^n}\sum_{\mathbf{a}\in\mathbb{Z}_M^n}|P_{\mathbf{a}}(n,k,h,m,\mathcal{T}^*)-P(m,\mathcal{T})|\leq mM^{1/2}|\mathcal{B}_{n,k,h}|^{-1/2}.$$

Proof. The attack $\mathcal{T}^*$ requests at each execution exactly m signatures. For each of those m signatures, the signature algorithm uses a pair of the form $(s_i, f(s_i))$ outputted by the generator, where s_i has values over a certain set S of keys. All these values $s_1,\ldots,s_m$ are pairwise independent. Therefore, the success probability of the attack $\mathcal{T}^*$ is equal to:

$$\sum_{(w_1,\ldots,w_m)\in S^m}\mathrm{Pr}_{\,\text{generator}}\,(\forall i, s_i=w_i)\times\mathrm{Pr}(\mathcal{T}^*\text{ succeeds}\mid\forall i, s_i=w_i).$$

Similarly, the success probability of the corresponding adaptive attack $\mathcal{T}$ against the original scheme is equal to:

$$\sum_{(w_1,\ldots,w_m)\in S^m}\mathrm{Pr}_{\,\text{oracle}}\,(\forall i, s_i=w_i)\times\mathrm{Pr}(\mathcal{T}\text{ succeeds}\mid\forall i, s_i=w_i),$$

with a uniform distribution for the $s_1,\ldots,s_m$. But the link between $\mathcal{A}_{\text{sig}}$ and $\mathcal{A}^*_{\text{sig}}$ ensures us that the conditional probabilities appearing in the two formulas are identical, and less than 1 by definition. Hence, the difference of the success probabilities of the two attacks $\mathcal{T}$ are $\mathcal{T}^*$ is bounded from the above by:

$$\sum_{(w_1,\ldots,w_m)\in S^m}\left|\mathrm{Pr}_{\,\text{generator}}\,(\forall i, s_i=w_i)-\mathrm{Pr}_{\,\text{oracle}}\,(\forall i, s_i=w_i)\right|.$$

This expression is equal to two times the statistical distance between the distribution obtained by sampling m outputs of the generator and the uniform distribution. Applying Lemma 3, the identity (4) and the bound (3) we conclude the proof. □

It follows that when the distribution of the generator output is sufficiently close to the uniform distribution, the **BPV** signature scheme is secure against existential forgeries, provided that the original scheme is.

6 Selecting the Parameters

First of all we remark that for $h = 2$, the estimate (4) and well known bounds of binomial coefficients imply that for every $\rho > 0$ and $k = \lfloor \rho \log M \rfloor$, for every $A > 0$ there exists a constant γ such that for $n = \lfloor \gamma \log M \rfloor$ we have

$$\frac{1}{M^n} \sum_{\mathbf{a} \in \mathbb{Z}_M^n} |P_{\mathbf{a}}(n, k, 2, m, \mathcal{T}^*) - P(m, \mathcal{T})| \leq m M^{-A+o(1)}. \tag{5}$$

For bigger (but still polynomial in $\log M$) values of n one can take even smaller values of k. For example, it is easy to see that for any $A > 0$ if one selects $n = \left\lfloor (A + 0.5) \log^2 M \right\rfloor$ and $k = \lfloor \log M / \log \log M \rfloor$ then

$$\frac{1}{M^n} \sum_{\mathbf{a} \in \mathbb{Z}_M^n} |P_{\mathbf{a}}(n, k, 2, m, \mathcal{T}^*) - P(m, \mathcal{T})| \leq m M^{-A+o(1)}. \tag{6}$$

In particular, the bounds (5) means that for any desired ratio $\rho \sim k / \log M$ there is a value of $n = O(\log M)$, that is, linear in the bit size of M such that the corresponding **BPV** generator signature scheme is secure against adaptive attacks (provided the original signature scheme is secure). Taking n quadratic in the bit-size of M, we derive (6) that growing speed-up can be achieved without compromising the security of the scheme.

We remark that case $h = 2$ has a very important advantage that the computation of $\mathbf{a} \cdot \mathbf{x}$, for $x \in \mathcal{B}_{n,k}$ does not require any storage, it takes only $k - 1$ multiplications.

Now we show that, selecting bigger values of h rather than $h = 2$, one can achieve even better results.

First of all we remark that if $n = k = \lfloor A \log M / \log \log M \rfloor$ and $h = \lceil n / \log n \rceil$ then

$$|\mathcal{B}_{n,k,h}| = M^{-A+o(1)},$$

thus the bound (6) holds for this selection of parameters as well. And the total computational cost is $n + o(n)$ thus asymptotically less than for repeated squaring and other exponentiation schemes without compromising on the security and with very reasonable storage requirements, namely with storing $k = O(\log M / \log \log M)$ precomputed values.

Some other choices of n, k and h are possible as well, depending of the particular application. For a given choice of n and a given number of multiplications (which is $k + h - 3$), one can compute the optimal values of k and

		Number of multiplications			
n	Comm. time	20	25	30	35
100	1.3s	2^{29} & 17	2^{17} & 21	2^{6} & 24	2^{-5} & 28
1000	13s	2^{-1} & 18	2^{-20} & 23	2^{-39} & 27	2^{-58} & 31
10000	2mins 11s	2^{-33} & 19	2^{-59} & 24	2^{-85} & 28	2^{-111} & 32
100000	22mins	2^{-66} & 20	2^{-99} & 24	2^{-132} & 29	2^{-166} & 33

Table 1: Value of $\sqrt{M/|\mathcal{B}_{n,k,h}|}$ and optimal value of k for a 160-bit M.

h that minimize the defect. Table 1 gives such values and the corresponding defect for a 160-bit M, together with the communication time, assuming a speed of 115200 bauds, the choice of a 1024-bit prime p, and a 160-bit hash function as in DSA (for n pairs and the hash tree with a 160-bit hash function, one needs to send $n(1.18) + 2n(0.16)$ Kbits).

7 Remarks

It is easy to see that one can obtain analogues of Theorem 4 for identification schemes as well.

One can also obtain similar results for the RSA-based generator with precomputation which has been introduced in [2] as well.

References

[1] M. Ajtai, 'Generating hard instances of lattice problems', *Electronic Colloq. on Comp. Compl.*, Univ. of Trier, **TR96-007** (1996), 1–29.

[2] V. Boyko, M. Peinado and R. Venkatesan, 'Speeding up discrete log and factoring based schemes via precomputations', *Proc. of Eurocrypt'98, Lect. Notes in Comp. Sci.*, Springer-Verlag, Berlin, **1403** (1998), 221–234.

[3] E. Brickell, D.M. Gordon, K.S. McCurley, and D. Wilson, 'Fast exponentiation with precomputation', *Proc. of Eurocrypt'92, Lect. Notes in Comp. Sci.*, Springer-Verlag, Berlin, **658** (1993), 200–207.

[4] E. F. Brickell and K. S. McCurley, 'An interactive identification scheme based on discrete logarithms and factoring', *Journal of Cryptology*, **5** (1992), 29–39.

[5] T. El Gamal, 'A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. Inform. Theory*, **31** (1985), 469–472.

[6] O. Goldreich, *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, Springer-Verlag, Berlin, 1999.

[7] D.M. Gordon, 'A survey of fast exponentiation methods, *Journal of Algorithms*, **27** (1998), 129–146.

[8] F. Griffin and I. E. Shparlinski, 'On the linear complexity of the Naor–Reingold pseudo-random function', *Proc. of 2nd Intern. Conf. on Inform. and Commun. Security, Sydney, 1999, Lect. Notes in Comp. Sci.*, Springer-Verlag, Berlin, 1999, 301–308.

[9] R. Impagliazzo and M. Naor, 'Efficient cryptographic schemes provably as secure as subset sum', *J. Cryptology*, **9** (1996), 199–216.

[10] C. H. Lim and P. J. Lee, 'More flexible exponentiation with precomputation', *Proc. of Crypto'94, Lect. Notes in Comp. Sci.*, Springer-Verlag, Berlin, **839** (1994), 95–107.

[11] M. Luby, *Pseudorandomness and cryptographic applications*, Princeton University Press, 19969.

[12] M. Naor and O. Reingold, 'Synthesizers and their application to the parallel construction of pseudo-random functions', *J. Comp. and Sys. Sci.*, **58** (1999), 336–375.

[13] National Institute of Standards and Technology (NIST), *FIPS Publication 186: Digital Signature Standard*, May 1994.

[14] P. Nguyen and J. Stern, 'The hardness of the hidden subset sum problem and its cryptographic implications', *Proc. of Crypto'99, Lect. Notes in Comp. Sci.*, Springer-Verlag, Berlin, **1666** (1999), 31–46.

[15] P. de Rooij, 'On the security of the Schnorr scheme using preprocessing', *Proc. of Eurocrypt'91, Lect. Notes in Comp. Sci.*, Springer-Verlag, Berlin, **547** (1991), 71–80.

[16] P. de Rooij, 'Efficient exponentiation using precomputation and vector addition chains', *Proc. of Eurocrypt'94, Lect. Notes in Comp. Sci.*, Springer-Verlag, Berlin, **950** (1995), 389–399.

[17] P. de Rooij, 'On Schnorr's preprocessing for digital signature schemes', *Journal of Cryptology*, **10** (1997), 1–16.

[18] C. P. Schnorr, 'Efficient identification and signatures for smart cards', *Proc. of Crypto'89, Lect. Notes in Comp. Sci.*, Springer-Verlag, Berlin, **435** (1990), 239–252.

[19] C. P. Schnorr, 'Efficient signature generation by smart cards', *Journal of Cryptology*, **4** (1991), 161–174.

[20] I. M. Vinogradov, *Elements of number theory*, Dover Publ., New York, 1954.