J. Cryptology (1997) 10: 1–16

Journal of
CRYPTOLOGY

On Schnorr's Preprocessing for Digital Signature Schemes*

Peter de Rooij**
PTT Research, P.O. Box 421,
2260 AK Leidschendam, The Netherlands

Communicated by Ivan B. Damgård

Received 29 April 1993 and revised 15 September 1995

Abstract. Schnorr's identification and signature schemes [10], [11] are efficient, discrete log-based protocols. Moreover, preprocessing algorithms are proposed that significantly speed up the computations of the prover (resp. signer). Therefore, this preprocessing greatly enhances the suitability for implementation on a smart card. The preprocessing algorithms can be used for other (discrete log-based) signature schemes as well.

The security of the preprocessing depends on a parameter k; the required storage is linear in k. In [10] and [11] the value $k = 8$ is suggested, for which the level of security is conjectured to be 2^{72} [11].

In this paper an attack on these preprocessing algorithms is presented. This attack retrieves the secret key in about $(k!)^2$ steps, using in the order of $\frac{1}{2}\sqrt{2\pi(k-1)k!}$ consecutive signatures or transcripts of identifications. For $k = 8$, this amounts to about 2^{31} steps and 700 signatures.

This attack is applicable to Brickell–McCurley, ElGamal, and DSS signatures as well, if the same preprocessing algorithm is used.

Key words. Cryptology, Cryptanalysis, Identification, Digital signature, ElGamal, DSS, Preprocessing, Smart card.

1. Introduction

At Crypto '89, a discrete log-based identification protocol and a related signature scheme especially suited for use on smart cards were proposed by Schnorr [10]. One optional feature of both was the possibility of using a preprocessing algorithm that substantially speeds up the prover's (or signer's) calculations. However, use of this preprocessing algorithm enables an attack that provides the prover's (signer's) secret key [9].

* Part of the results of this paper were presented at Eurocrypt '91 [9].

** Currently, the author is at Europay International, Chaussée de Tervuren 198A, 1410 Waterloo, Belgium.

An improved preprocessing algorithm, resistant to this attack, was proposed in [11]. Below, the attack is adapted to the new preprocessing algorithm. This new attack still is applicable to the old preprocessing, and is substantially faster than the old attack from [9].

Since the preprocessing algorithms can be used in any protocol where a random power of a fixed base is to be computed, the scope of this paper is wider than just the application to Schnorr's protocols from [10] and [11]. The proposed attack can be adapted to any protocol where Schnorr's preprocessing is applied, provided that it satisfies a (rather mild) assumption. Specifically, it is shown to be applicable to Brickell–McCurley, ElGamal, and DSS signatures [1], [4], [7] with Schnorr's preprocessing.

This paper is organized as follows. Section 2 briefly describes Schnorr's identification protocol and signature scheme, and gives the preprocessing algorithm. The new attack mentioned above is given in Section 3. Section 4 is dedicated to the computation of the complexity of this attack and the required number of signatures. In Section 5 the attack is generalized in several ways. The consequences for the complexity of the attack of changing the preprocessing and of lifting some of the assumptions are discussed, and a straightforward adaption of the attack to ElGamal and DSS signatures with the same preprocessing is given. Section 6, finally, contains the conclusions.

2. Description of the Algorithm

2.1. *Background*

In [11] a discrete log-based identification protocol and a related signature scheme are proposed. The identification protocol is based on the Chaum–Evertse–van de Graaf protocol [2]. Essentially, it condenses this protocol into one single round. The signature scheme is an extension of the identification protocol. We briefly describe the identification protocol and the signature scheme. This provides the necessary background for the introduction of the preprocessing algorithm and its cryptanalysis. For more details, see [11].

2.1.1. *Preliminaries*

The following parameters are chosen once and for all, and are known to all users: a large prime p, a prime q that divides $p-1$, a primitive qth root of unity $\alpha \in \mathbb{Z}_p$, and a security parameter t. In [11] it is proposed taking p and q in the order of 512 bits and 140 bits respectively, and $t = 72$.

Each user $\mathcal{A}$ chooses a secret key $s_{\mathcal{A}} \in \mathbb{Z}_q^*$. The corresponding public key is $v_{\mathcal{A}} = \alpha^{-s_{\mathcal{A}}} \bmod p$. Of course, the authenticity of this public key must be checked by the verifier. This is done by means of a *certificate*. However, since this aspect is of no consequence in this paper, it is ignored from now on.

In the rest of this paper the index $\mathcal{A}$ is used only where necessary to avoid confusion.

2.1.2. *The Identification Protocol*

Suppose *prover* $\mathcal{A}$ (Alice) wants to prove her identity to *verifier* $\mathcal{B}$ (Bob). First, $\mathcal{A}$ picks a random number $r \in \mathbb{Z}_q^*$ and sends the *initial commitment* $x = \alpha^r \bmod p$ to $\mathcal{B}$. Then $\mathcal{B}$ returns a random number $e \in \{0, \ldots, 2^t - 1\}$, called the *challenge*, to $\mathcal{A}$. Next, $\mathcal{A}$

sends $y = r + s_{\mathcal{A}} \cdot e \bmod q$ to $\mathcal{B}$. Finally, $\mathcal{B}$ checks $\mathcal{A}$'s proof of identity by calculating $\bar{x} = \alpha^y v_{\mathcal{A}}^e \bmod p$. The proof will be accepted if and only if $\bar{x} = x$.

In what follows all calculations are performed modulo q, except where indicated otherwise.

2.1.3. *The Signature Scheme*

The identification protocol is extended to a signature scheme analogous to the extensions of the Fiat–Shamir and Guillou–Quisquater identification protocols [5], [8]. That is, a t-bit hash value of the initial commitment x and the message m to be signed replaces the challenge. The signature consists of this hash value and of y as in the identification protocol.

Let h denote the hash function that is used to compute the hash value. A message m is signed by *signer* $\mathcal{A}$ as follows. First, $\mathcal{A}$ picks a random number $r \in \mathbb{Z}_q^*$ and calculates $x = \alpha^r \bmod p$; from this $e = h(x, m)$ and $y = r + s_{\mathcal{A}} \cdot e$ are computed. The signature consists of the pair (y, e). In what follows a pair (y, e) is called a *signature*, even if it is constructed in the course of an instance of the identification protocol.

The signature of $\mathcal{A}$ on m can be checked by computing $\bar{x} = \alpha^y v_{\mathcal{A}}^e \bmod p$ and $\bar{e} = h(\bar{x}, m)$. The signature will be accepted if and only if $\bar{e} = e$.

2.2. *The Preprocessing Algorithm*

Essentially, the preprocessing algorithm from [11] is a modified version of the algorithm from [10]. The purpose of both is the reduction of the computational effort of the prover/signer. This effort is determined by the effort of performing the modular exponentiation $\alpha^r \bmod p$. The preprocessing essentially reduces this effort to that of a few multiplications only. This is achieved by taking, instead of a random r, a linear combination of several independently and randomly chosen numbers $r_i \in \mathbb{Z}_q^*$ for which $x_i = \alpha^{r_i} \bmod p$ is precomputed.

For this purpose, each user initially stores a collection of k such pairs (r_i, x_i), $0 \leq i < k$. Here k is a security parameter; the proposed value [10], [11] is $k = 8$. Then, for each signature, the pair (r, x) is chosen as a combination of the stored pairs (r_i, x_i).

Subsequently, the collection of pairs is "rejuvenated" by replacing one of the pairs (r_i, x_i) by a similar combination. More precisely, the first time, (r_0, x_0) is replaced by a combination of the pairs (r_i, x_i) for $0 \leq i < k$. This new pair is denoted (r_k, x_k). The next time, (r_1, x_1) is replaced by (r_{k+1}, x_{k+1}), and so on.

More specifically, a new pair (r_{t+k}, x_{t+k}) is computed from the stored pairs (r_{t+i}, x_{t+i}) for $0 \leq i < k$ in both [10] and [11] as $(\sum_j 2^j r_{t+a(j)}, \prod_j x_{t+a(j)}^{2^j})$. Here the $a(j)$ select one of the stored pairs. The pair (r, x) is an intermediate result in the computation of (r_i, x_i).

In [10] the $a(j)$ were randomly selected from $\{0, \ldots, k-1\}$ and a security parameter d determined the number of pairs used in these combinations. In the new preprocessing algorithm [11], the $a(j)$ for $0 \leq j < k$ represent a random permutation of $\{0, \ldots, k-1\}$, so that all k stored pairs are used.

The index of the new pair can be used as a sequential number for the signature. Denote this sequential number by i, the value of r used in the corresponding initial commitment by r_i^*, and the initial commitment itself by x_i^*. The signature itself is (y_i, e_i), and the pair (r_{i-k}, x_{i-k}) is replaced by (r_i, x_i), using $a_i(\cdot)$ for the selection of the stored pairs.

Finally, let $\mathcal{S}_k$ denote the symmetric group of permutations on k elements (here on the integers $0, 1, \ldots, k-1$).

(The notation here is different from that used in [11]. Clearly, this does not alter the preprocessing itself.)

The preprocessing algorithm [11] is as follows. (The preprocessing from [10] differs only in the form and number of terms $a_i(j)$.)

```
The Preprocessing Algorithm
Initialization. Load $k$ pairs $(r_i, x_i)$ as above, $0 \le i < k$;
    $i := k$;
1. Pick a random permutation $(a_i(0), \ldots, a_i(k-1)) \in \mathcal{S}_k$;
   $a_i(k) := 0$; $a_i(k+1) := k-1$;
2. $r_i^* := r_{i-k} + 2r_{i-1} \bmod q$; $r_i := \sum_{j=0}^{k+1} 2^j r_{a_i(j)+i-k} \bmod q$;
   $x_i^* := x_{i-k} \cdot x_{i-1}^2 \bmod p$; $x_i := \prod_{j=0}^{k+1} (x_{a_i(j)+i-k})^{2^j} \bmod p$;
3. Keep the pair $(r_i^*, x_i^*)$ ready for the next signature;
   Keep the pairs $(r_j, x_j)$, $i-k+1 \le j \le i$, stored;
   (i.e., replace $(r_{i-k}, x_{i-k})$ with $(r_i, x_i)$)
4. $i := i + 1$;
   goto 1 for the next signature.
```

This preprocessing requires a storage of k pairs (r_i, x_i) and a computational effort of $2k+2$ multiplications modulo p, see [11].

3. A New Attack

3.1. *Introduction*

In the preprocessing algorithm from [10] it was possible that only two of the stored pairs (r_i, x_i) were used for the generation of the new pair. This introduced possible dependencies of only three signatures that were sufficient to retrieve the secret key. This was exploited by the attack from [9].

The main difference between the old preprocessing algorithm [10] and the new version [11] is the fact that in the new version it is guaranteed that all stored pairs are used. Below, it is shown that nevertheless it is possible to find signatures that contain dependencies that yield the secret key of a user.

3.2. *Dependencies Between Signatures*

In this section we generalize the attack from [9] to the new preprocessing. Assume that p, q, α, t, and some number of successive signatures are available to the enemy. Then, giving y and e in the ith signature an index i, the following equations hold:

$$r_i^* = r_{i-k} + 2r_{i-1}, \qquad i = k, k+1, \ldots, \tag{1}$$

$$r_i = \sum_{j=0}^{k+1} 2^j r_{a_i(j)+i-k}, \qquad i = k, k+1, \ldots, \tag{2}$$

$$y_i = r_i^* + se_i, \qquad i = k, k+1, \ldots. \tag{3}$$

This is exactly the same as in [10] except for the choice and number of $a_i(j)$'s. Therefore the *linking equations* can be derived in exactly the same manner as in [9]. That is, from (1) and (3) it follows that $r_{i-1} = \frac{1}{2}(y_i - se_i - r_{i-k})$ for all $i \geq k$. Repeated substitution using this equality yields, for $j \geq 0$, $c \geq 1$,

$$r_{j+c(k-1)} = (-\tfrac{1}{2})^c r_j - \sum_{l=1}^{c} (-\tfrac{1}{2})^{c-l+1} (y_{j+l(k-1)+1} - se_{j+l(k-1)+1}).$$

This equation is called a *linking equation*. Note that it is vacuously satisfied for $c = 0$ as well. Below, this equation is used for $j = i \bmod (k-1)$ and $c = i \operatorname{div}(k-1)$.

We can use this linking equation to write any r_i with $i \geq 0$ as a function of $r_{i \bmod (k-1)}$, s and the signatures (y_j, e_j) for $j = i \bmod (k-1) + k, i \bmod (k-1) + 2k - 1, \ldots, i+1$. If those signatures are known, then this is a *linear combination* of $r_{i \bmod (k-1)}$ and s with known constant and known coefficients. We can write this equation as

$$r_i = (-\tfrac{1}{2})^{i \operatorname{div}(k-1)} r_{i \bmod (k-1)} + Y_i - sE_i, \tag{4}$$

where

$$E_i = -\sum_{l=1}^{i \operatorname{div}(k-1)} (-\tfrac{1}{2})^{i \operatorname{div}(k-1)-l+1} e_{i_{\bmod}(k-1)+l(k-1)+1} \tag{5}$$

and similarly for Y_i

This fact can be used to rewrite any equation of the same form as (2) to a linear equation in $r_0, \ldots, r_{k-2}$ and s. To see this, first note that (2) is a linear equation in $r_{i-k}, r_{i-k+1}, \ldots, r_i$. By the form of $a_i(j)$, it can be rewritten to

$$r_i = 2^k r_{i-k} + 2^{k+1} r_{i-1} + \sum_{j=0}^{k-1} A_i(j) r_{i-k+j},$$

where $A_i(j) = \sum_{a_i(l)=j} 2^l$, where the summation ranges over the terms with $0 \leq l < k$ only. Since in the new preprocessing $(a_i(0), \ldots, a_i(k-1))$ represents a permutation for all i, each sum consists of exactly one term, and $\log_2 A_i(\cdot)$ represents the inverse of this permutation. (A_i itself is a permutation on the symbols $1, 2, \ldots, 2^{k-1}$.)

Substitution of (4) and reordering of terms yields

$$\begin{aligned} &Y_i - 2^k Y_{i-k} - 2^{k+1} Y_{i-1} - \sum_{j=0}^{k-1} A_i(j) Y_{i-k+j} \\ &\quad = -(-\tfrac{1}{2})^{i \operatorname{div}(k-1)} r_{i \bmod (k-1)} + \sum_{j=0}^{k-1} A_i(j) (-\tfrac{1}{2})^{(i-k+j) \operatorname{div}(k-1)} r_{(i-k+j) \bmod (k-1)} \\ &\qquad + s\left(E_i - 2^k E_{i-k} - 2^{k+1} E_{i-1} - \sum_{j=0}^{k-1} A_i(j) E_{i-k+j}\right), \end{aligned} \tag{6}$$

which obviously is a linear equation in r_0 up to r_{k-2} and s, with coefficients depending on i, on $A_i(\cdot)$, and on (the signatures required to compute) Y_{i-k} up to Y_i and E_{i-k} up to E_i. Clearly, this equation can be written in the form

$$\sum_{j=0}^{k-2} \mathcal{M}_{ij} r_j + s\mathcal{E}_i = \mathcal{Y}_i, \tag{7}$$

where $\mathcal{Y}_i$ and $\mathcal{E}_i$ depend on i, on $A_i(\cdot)$, and some signatures only. It easily follows from (6) that, for all j, the coefficients $\mathcal{M}_{ij}$ depend on i and $A_i(\cdot)$ only. This is crucial to the attack in [9] as well as to the one presented here.

It follows that, given n different equations of the same form as (2), say for the sequential numbers $i_0 < i_1 < \cdots < i_{n-1}$, the following set of equations can be formed:

$$\left[\begin{array}{c|c} M & \begin{matrix}\mathcal{E}_{i_0}\\ \vdots \\ \mathcal{E}_{i_{n-1}}\end{matrix}\end{array}\right] \cdot \left[\begin{array}{c} r_0 \\ \vdots \\ r_{k-2} \\ \hline s \end{array}\right] = \left[\begin{array}{c} \mathcal{Y}_{i_0} \\ \vdots \\ \mathcal{Y}_{i_{n-1}} \end{array}\right]. \tag{8}$$

Here M is an $n \times (k-1)$ matrix with entries $M_{lj} = \mathcal{M}_{i_l j}$ as in (7) with index $i = i_l$. For details, see Appendix A.

3.3. *Use of the Dependencies*

Up to now, only the *form* of (1)–(3) is used, so everything that is derived holds for both versions of the preprocessing.

In the old preprocessing it is possible that only two $A_i(j)$'s are nonzero. That is, it is possible that a row of M has a support of weight 2, see Appendix A. The attack described in [9] finds a matrix M with three rows with the same support of weight 2 and with $i_0 \equiv i_1 \equiv i_2 (\bmod k-1)$. Then (8) reduces to a system of three equations in three unknowns, which can be solved with high probability. The solution provides an estimate $\bar{s}$ for s, which is correct if and only if $\alpha^{-\bar{s}} = v$. The search for this matrix M is performed as an exhaustive search over all possible candidate matrices; i.e., over all suitable triples (i_0, i_1, i_2) and all suitable $A_{i_j}(\cdot)$ for which the required signatures are available. For details, see [9]. (The formulation is different there.)

That attack will not work here, as it is guaranteed by the choice of the $a_i(l)$'s that all rows of M have support of weight $k-1$, see Appendix A. This implies that the attack will require a $k \times (k-1)$ matrix M. The work factor then increases to about $(k!)^k$ steps, as there are that many different possible M-matrices. This is infeasible for the proposed parameter $k = 8$.

The attack presented here will find two *identical* rows of M, say with indices i_0 and i_1. Given that, the entries of M in (7) cancel out by subtraction of row i_1 from i_0 of this system. This yields

$$s(\mathcal{E}_{i_0} - \mathcal{E}_{i_1}) = \mathcal{Y}_{i_0} - \mathcal{Y}_{i_1}, \tag{9}$$

from which s is found if $\mathcal{E}_{i_0} - \mathcal{E}_{i_1} \neq 0$.

From the form of the entries of the matrix M (see Appendix A) it can easily be seen that two rows of M are identical if and only if $A_{i_0} = A_{i_1}$ and $i_0 \equiv i_1 (\bmod k-1)$, say

$i_j = \lambda + \mu_j(k-1)$, with $0 \le \lambda < k-1$ and $\mu_j > 0$. In that case, denoting $\mu = \mu_1 - \mu_0$,

$$\begin{aligned}\mathcal{E}_{i_0} - \mathcal{E}_{i_1} &= (-2)^{\mu_0} 2^k((-2)^{\mu} e_{i_1} - e_{i_0}) \\ &\quad + \sum_{l=1}^{\mu} (-2)^{\mu_0+l-1} \left(e_{i_0+l(k-1)+1} - \sum_{j=0}^{k-1} A_{i_0}(j) e_{i_0+j+(l-1)(k-1)} \right). \end{aligned} \quad (10)$$

The same expression with all e_*'s replaced by the corresponding y_*'s holds for $\mathcal{Y}_{i_0} - \mathcal{Y}_{i_1}$. See Appendix B for details on the computation.

This shows that in order to compute s from (9), we only need the $\mu(k-1)+2$ signatures with indices i_0 through $i_1 + 1$, and the correct value of A_{i_0}.

The difference $\mathcal{E}_{i_0} - \mathcal{E}_{i_1}$ is nonzero with overwhelming probability. This can be seen as follows. Given all e_i's that occur in (10), except one, there is at most one value for this last one that forces $\mathcal{E}_{i_0} - \mathcal{E}_{i_1} = 0$. This happens with negligible probability, since for identification, e is a random t-bit value (proposed: $t = 72$); for signing, it is the output of a hash function.

We may conclude that, for any i_0, any $i_1 \equiv i_0 (\mathrm{mod}\, k-1)$, $i_1 > i_0$, and any A_{i_0}, (9) provides an estimate for s with overwhelming probability. This estimate is correct if $A_{i_1} = A_{i_0}$.

If an adversary obtains the set of signatures $\{(y_i, e_i) | i_0 \le i \le i_1 + 1\}$ and if $i_0 \equiv i_1 (\mathrm{mod}\, k-1)$, $i_0 < i_1$, we say that he has a *candidate* for finding s. We denote this candidate by $\langle i_0, i_1 \rangle$.

Given a candidate, he may just assume that $A_{i_0} = A_{i_1}$ and apply (9) for all $k!$ possible value of A_{i_0}. Each of these instances of (9) provides an estimate for s with overwhelming probability. The correctness of an estimate $\bar{s}$ can easily be assessed by checking whether $\alpha^{-\bar{s}}$ equals the public key v.

If indeed $A_{i_0} = A_{i_1}$, he will eventually hit the correct value of A_{i_0} and find s (with overwhelming probability). This will take at most $k!$ steps, where each step consists of the computation of an estimate $\bar{s}$ by (9) for some choice of A_{i_0} and the verification $\alpha^{-\bar{s}} \overset{?}{=} v$. The computational effort of a step is determined by this modular exponentiation.

If $A_{i_0} \ne A_{i_1}$, he will find incorrect estimates in all $k!$ steps. (We neglect the probability that an incorrect system (8) will nevertheless yield the correct solution s.)

So, given a candidate, the enemy performs at most $k!$ steps, yielding either s or "failure." This whole process is called *checking a candidate*.

A candidate is *good* if $A_{i_0} = A_{i_1}$, which happens with probability $1/|\mathcal{S}_k| = 1/k!$. Recall that checking a good candidate provides s with overwhelming probability.

Below, we assume that a good candidate always provides s. That is, we neglect the probability that a good candidate is found that does not provide s. This seems reasonable, as this happens with probability at most $2^{-t} (= 2^{-72})$. (We may also define the probability of success of the proposed attack as the probability that a good candidate provides s.)

3.4. *An Attack*

Suppose the enemy possesses a number of consecutive signatures. He can partition the sequential numbers into $k-1$ sets Υ_λ, $0 \le \lambda < k-1$, according to their residues λ modulo $(k-1)$. Then, for each pair i_0, i_1 in one of the Υ_λ's, the set of signatures $\langle i_0, i_1 \rangle = \{(y_i, e_i) | i_0 \le i \le i_1 + 1\}$ is a candidate; we say that this candidate is in Υ_λ.

Given this, an attack could proceed by checking all those candidates in some order. For example, this can be done as follows. For all values $\lambda \in \{0, \dots, k-2\}$ check each candidate in Υ_λ, until s is found. This is just an exhaustive search over all possible candidates possessed.

4. The Complexity of the Attack

4.1. *Introduction*

In this section the expected required number of signatures and the expected number of steps for the attack described above are computed. First, we calculate the expected sufficient size of the Υ_λ's as introduced in the previous section. From this, both expectations can be computed.

First, observe that the probability that a candidate is good is $1/k!$. Therefore, it is intuitively expected that the number of required signatures be in the order of $\sqrt{k!}$, by the birthday paradox. Since the number of steps of the attack equals the number of estimates for s, it would then be expected that the number of steps be in the order of $(k!)^2$, as there are $k!$ estimates per candidate, and in the order of $k!$ candidates in a pool of signatures of size $\sim \sqrt{k!}$. These intuitions prove accurate.

4.2. *The Expected Number of Required Signatures*

Recall that a candidate provides s with probability $1/k!$. Therefore, given a collection of candidates $\Upsilon = \{i_{\min}, i_{\min} + k - 1, \dots, i_{\min} + (N-1)(k-1)\}$ there is a probability

$$\Pr\{\text{no success} \mid \Upsilon\} = \prod_{i=1}^{N-1} \left(1 - \frac{i}{k!}\right)$$

that no candidate provides s. Therefore, the probability that the above attack requires at least $(k-1)N + 1$ consecutive signatures to obtain a good candidate, providing s, given by

$$\Pr\{\mathbf{N} \geq N\} = \prod_{i=1}^{N-2} \left(1 - \frac{i}{k!}\right)^{k-1}, \tag{11}$$

where $\mathbf{N}$ is the random variable denoting the required cardinality of the sets Υ_λ. From this it follows that the expected value of $\mathbf{N}$ amounts to

$$\mathrm{E}_{\mathbf{N}}(\mathbf{N}) = \sum_{N=1}^{\infty} N \Pr\{\mathbf{N} = N\} \approx \sqrt{\frac{\pi k!}{2(k-1)}}, \tag{12}$$

see Appendix C. The required number of signatures is $(k-1)E_{\mathbf{N}}(\mathbf{N}) + 1$, or about $\sqrt{\frac{1}{2}\pi(k-1)k!}$. For $k = 8$, this amounts to 666 consecutive signatures.

4.3. *The Complexity of the Attack*

The expected number of candidates, $\mathrm{E}_{\mathcal{N}}$ (# candidates), to check for each value of λ follows from

$$\mathrm{E}_{\mathcal{N}}(\text{\# candidates}) = \sum_{N=1}^{\infty} \binom{N}{2} \Pr\{\mathcal{N} = N\}. \tag{13}$$

This yields

$$\frac{k!}{k-1} - 1 < \mathrm{E}_{\mathcal{N}}(\text{\# candidates}) < 2 + \frac{k!}{k-1} + \sqrt{\frac{\pi k!}{2(k-1)}}.$$

See Appendix D. For $k = 8$, this implies $\mathrm{E}_{\mathcal{N}}(\text{\# candidates}) < 5857$.

From this we find the expected workload WL: the above number of candidates must be checked for each value of λ; checking each candidate requires $k!$ steps. That is,

$$\mathrm{WL} = (k-1) \cdot \mathrm{E}_{\mathcal{N}}(\text{\# candidates}) \cdot k! \approx (k!)^2,$$

which is $1.6 \cdot 10^9 \approx 2^{31}$ for $k = 8$.

5. Generalization

5.1. *Other Preprocessing Algorithms*

The attack will work for any preprocessing algorithm where (1) and (3) hold. The expected number of signatures is approximately

$$\sqrt{\tfrac{1}{2}\pi(k-1)\mathcal{N}}$$

and the workload amounts to approximately $\mathcal{N}^2$ steps, if the analogue if (2) allows $\mathcal{N}$ possibilities.

For example, in the described attack we have $\mathcal{N} = k!$. In the preprocessing from [10] we have $\mathcal{N} = k^{d-2}$. For $k = 8$ and $d = 6$, the suggested values of the security parameters, the number of signatures there is about 212; the workload then is $(8^4)^2 = 2^{24} = 1.7 \cdot 10^7$. (The attack from [9] required 2000 signatures with a workload of 2^{38}.)

5.2. *Relaxing the Assumptions*

The attack described above requires a relatively small number of signatures in order to be successful, but those signatures should be *consecutive* signatures by the same entity. This requirement can be relaxed, at the expense of increasing the number of required signatures. The workload remains the same, as is shown below.

Observe that any $k + 1$ consecutive signatures form a candidate, so the probability of recovering the corresponding secret key s from those signatures is $1/k!$. Moreover, for N such *minimal candidates* $\langle i_0, i_0 + k - 1\rangle, \ldots, \langle i_{N-1}, i_{N-1} + k - 1\rangle$, the probabilities of success are independent.

Therefore, the expected required number of minimal candidates is $k!$. Note that these minimal candidates need not even be related to the same signer/prover.

The complexity of this attack (checking minimal candidates) is of the same order as the attack given above, namely $(k!)^2$; only the number of required signatures is $(k+1)!$, instead of $\frac{1}{2}\sqrt{2\pi(k-1)k!}$. For $k = 8$, this is a little over 360,000, instead of 666.

Sets of $2k - 1$ consecutive signatures may also be used. Such a set provides $k - 1$ independent minimal candidates, namely one for each index modulo $k - 1$. Again, N disjunct sets of $2k - 1$ signatures provide $N(k - 1)$ independent candidates. That is, expected value of N is $k!/(k - 1)$. We see that $k!/(k - 1)$ disjunct sets of $2k - 1$ consecutive signatures (in total $5760 \cdot 15 = 86{,}400$ signatures for $k = 8$) are sufficient as well. The workload is still the same.

The significantly lower number of signatures required for the proposed attack is a consequence of the birthday paradox. Since the expected workload WL is in this case not determined by the number of signatures, but by the number of checks, the birthday paradox does not help in reducing the complexity.

5.3. *Adaption to Other Signature Schemes*

The preprocessing algorithm can be applied in any situation where random powers of fixed bases are to be computed. This will occur in most discrete log-based identification protocols and signature schemes. However, the preprocessing will in general not be applicable in key exchange protocols, as either the exponent or the base is chosen by the other party. For example, in the Diffie–Hellman key exchange, the other party chooses the exponent [3]; in Günther's key exchange protocol the base is chosen by the other party [6].

If the preprocessing discussed here is used in other signature schemes, (3) will in general be different. If the resulting equation is linear in r^* (i.e., its coefficient is known), it is possible to adapt the attack. We give three examples.

5.3.1. *Brickell–McCurley*

The identification scheme and the related signature scheme by Brickell and McCurley [1] are variants of Schnorr's schemes. The difference lies in the fact that all reductions modulo q are replaced by reductions modulo $p - 1$, while q is kept secret. This makes no difference for the proposed attack, except that all computations now must be performed modulo $p - 1$ instead of modulo q.

An additional issue is invertibility modulo $p - 1$. First, note that 2 must be invertible for the computation of $\mathcal{E}_*$ and $\mathcal{Y}_*$ in (9), see (15). However, 2 is never invertible modulo $p - 1$. Since the product R of all small factors of $p - 1$ can be determined easily, this problem can be avoided by performing the computations modulo $(p - 1)/R$, rather than modulo $p - 1$. This yields s modulo $(p - 1)/R$. Since this fixes s modulo q as well, this provides an equivalent key.

Next, note that $\mathcal{E}_{i_0} - \mathcal{E}_{i_1}$ need not be invertible modulo $(p - 1)/R$. This happens with probability inversely proportional to the smallest factor of $(p - 1)/R$. That is, as before, this probability is negligible.

So, if Schnorr's preprocessing is used in the Brickell–McCurley identification or signatures, the proposed attack can be applied. The complexity and required number of signatures are the same. The exponentiation that determines the workload of checking a candidate takes about $|p-1|/|qR|$ times as long as for Schnorr's signature scheme, as the complexity of exponentiation is linear in the length of the exponent.

5.3.2. *ElGamal*

For ElGamal's signature scheme [4], a signature on a message m is a pair (ρ, σ), with $\rho = \alpha^r$, that satisfies

$$m \equiv s\rho + r\sigma \pmod{p-1}.$$

(The notation of [4] is changed to conform to the notation in this paper. Also, note that the public key is α^s, not α^{-s}. Obviously, this makes no difference here.) It follows that (3) can be replaced by

$$\frac{m_i}{\sigma_i} = r_i^* + s\frac{\rho_i}{\sigma_i},$$

with all computations modulo $p-1$. Therefore, with $y_i = m_i/\sigma_i$ and $e_i = \rho_i/\sigma_i$ for all i we obtain the same equations as for Schnorr's signature scheme. Note that indeed both y_i and e_i can be computed by anyone in possession of the corresponding signature.

This implies that the attack will work for ElGamal signatures with Schnorr's preprocessing as well. The complexity and required number of signatures are the same, but all computations are modulo $(p-1)/R$, as for Brickell–McCurley. Since the verification still determines the workload of checking a candidate, the only computational difference is the length of the exponent. Thus, the attack typically takes about four times as long as for Schnorr's signature scheme.

5.3.3. *DSS*

For the proposed Digital Signature Standard DSS [7], a signature on a message m is a pair (ρ, σ), that satisfies

$$\rho = (\alpha^r \bmod p) \bmod q \quad \text{and} \quad \sigma = (r^{-1}(h(m) + s\rho)) \bmod q.$$

(The notation of [7] is changed to conform to the notation in this paper. Again, the public key is α^s, not α^{-s}, which again makes no difference.) From this it follows that

$$r = \sigma^{-1}(h(m) + s\rho) \bmod q.$$

In this case, (3) can be replaced by

$$\sigma_i^{-1}h(m_i) = r_i^* - s\sigma_i^{-1}\rho_i.$$

Therefore, with $y_i = \sigma_i^{-1}h(m_i)$ and $e_i = -\sigma_i^{-1}\rho_i$ we obtain the same equations as for Schnorr's signature scheme. Note that indeed both y_i and e_i can be computed by anyone in possession of the corresponding signature.

This implies that the attack will work for DSS signatures with Schnorr's preprocessing as well. The complexity and required number of signatures are the same as for Schnorr's signature scheme.

6. Discussion and Conclusions

The idea of preprocessing the computation of a random power of a fixed base is interesting, as it reduces the effort to only a few multiplications. This is especially worthwhile for smart cards. Care must be taken, however, that no too much information leaks: the replacement of a "true random number" by a pseudorandom number always provides side information.

The preprocessing algorithm as proposed in [11] is not sufficiently secure, since the attack described above retrieves the secret key in about $(k!)^2$ steps, using in the order of $\frac{1}{2}\sqrt{2\pi(k-1)k!}$ consecutive signatures or transcripts of identifications, where k is a security parameter. For the proposed value $k = 8$, this amounts to about 2^{31} steps and 666 signatures. The workload for each step is determined by a modular exponentiation (for Schnorr's signature scheme and DSS with a 140-bit and 160-bit exponent, respectively).

For the preprocessing algorithm from [10], the secret key can be retrieved in $k^{2(d-2)}$ steps, using about $\frac{1}{2}\sqrt{2\pi(k-1)k^{d-2}}$ consecutive signatures, where d is an additional security parameter. For the proposed values $k = 8$ and $d = 6$ this amounts to 2^{24} steps and 212 signatures.

The attack does not seem to depend very heavily on the underlying signature scheme, as it only requires a linear equation in the random exponent. A Schnorr, Brickell–McCurley, ElGamal, or DSS signature provides such an equation, as it must hold for a successful verification.

Furthermore, the attack shows that it is not sufficient to require statistical independence of the exponents of any k consecutive signatures, where k is the number of stored pairs. The rationale for this requirement was that at least $k + 1$ consecutive signatures would be required, or some number of nonconsecutive ones.

This indeed holds for the proposed attack. However, only two of the random choices involved in those signatures are of consequence for the attack. It is the number of possibilities for these choices that determines the workload, not the number of signatures.

The attack works because of the extra information provided by the signature itself, namely (3), which is linear in r^* (with a known coefficient). Together with the way of generating r^* (equation (1)) this is sufficient to reduce the number of involved random choices to two.

Acknowledgments

The author would like to thank Johan van Tilburg, Arjen Lenstra, Claus Schnorr, Jean-Paul Boly, and the referees for their comments and for the fruitful discussions about this paper, and Yacov Yacobi for pointing out the issue of adapting the attack to other signature schemes.

Appendix A. The Form of the M-Matrix

In this appendix the exact form of (7) is derived. Let

$$\mu = i \operatorname{div}(k-1) \qquad \text{and} \qquad \lambda = i \bmod (k-1).$$

Then

$$(i-k+j)\operatorname{div}(k-1)=\begin{cases}\mu-2 & \text{if } j=\lambda=0,\\ \mu-1 & \text{if } 1-\lambda\le j<k-\lambda,\\ \mu & \text{if } k-\lambda\le j\le k-1\end{cases}$$

and

$$(i-k+j)\operatorname{mod}(k-1)=\begin{cases}k-2 & \text{if } j=\lambda=0,\\ \lambda+j-1 & \text{if } 1-\lambda\le j<k-\lambda,\\ \lambda+j-k & \text{if } k-\lambda\le j\le k-1.\end{cases}$$

Multiplication of (6) by $(-2)^\mu$ then yields

$$\begin{aligned}(-2)^\mu&\left(Y_i-2^kY_{i-k}-2^{k+1}Y_{i-1}-\sum_{j=0}^{k-1}A_i(j)Y_{i-k+j}\right)\\ &=-r_\lambda+4\sum_{j=\lambda}^{0}A_i(j)r_{k-2}-2\sum_{j=1-\lambda}^{k-1-\lambda}A_i(j)r_{\lambda+j-1}+\sum_{j=k-\lambda}^{k-1}A_i(j)r_{\lambda+j-k}\\ &\quad+s(-2)^\mu\left(E_i-2^kE_{i-k}-2^{k+1}E_{i-1}-\sum_{j=0}^{k-1}A_i(j)E_{i-k+j}\right),\end{aligned}\tag{14}$$

where, in all summations, $0\le j\le k-1$. Setting $\mathcal{M}_{ij}$ to the coefficient of r_j in this equation, and

$$\mathcal{E}_i=(-2)^\mu\left(E_i-2^kE_{i-k}-2^{k+1}E_{i-1}-\sum_{j=0}^{k-1}A_i(j)E_{i-k+j}\right),\tag{15}$$

and similarly for $\mathcal{Y}_i$, this provides (7). Note that indeed the coefficients $\mathcal{M}_{ij}$ depend on A_i and on i (or even on $i \bmod (k-1)$) only; $\mathcal{Y}_i$ and $\mathcal{E}_i$ depend on A_i, on i, and a number of signatures.

Finally, (14) enables us to give an explicit form of a row $M_{i\cdot}$ of M (suppressing the indices i):

$$\begin{array}{cccccll}-2A(1)-1 & -2A(2) & \cdots & -2A(k-2) & -2A(k-1)+4A(0) & \text{if} & \lambda=0,\\ A(k-1)-2A(0) & -2A(1)-1 & \cdots & -2A(k-3) & -2A(k-2) & \text{if} & \lambda=1,\\ A(k-2) & A(k-1)-2A(0) & \cdots & -2A(k-4) & -2A(k-3) & \text{if} & \lambda=2,\\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots\\ A(2) & A(3) & \cdots & A(k-1)-2A(0) & -2A(1)-1 & \text{if} & \lambda=k-2.\end{array}$$

Appendix B. Computation of $\mathcal{E}_{i_0} - \mathcal{E}_{i_1}$

Let $i_t = \lambda + \mu_t(k-1)$ for $t = 0, 1$. Furthermore, let $\mu = \mu_1 - \mu_0$. According to (5) it holds that

$$
\begin{aligned}
&(-2)^{\mu_1} E_{j+\mu(k-1)} - (-2)^{\mu_0} E_j \\
&\quad = -(-2)^{\mu_1} \sum_{l=1}^{j\,\mathrm{div}(k-1)+\mu} (-\tfrac{1}{2})^{j\,\mathrm{div}(k-1)-l+1+\mu} e_{j\,\mathrm{mod}(k-1)+l(k-1)+1} \\
&\qquad + (-2)^{\mu_0} \sum_{l=1}^{j\,\mathrm{div}(k-1)} (-\tfrac{1}{2})^{j\,\mathrm{div}(k-1)-l+1} e_{j\,\mathrm{mod}(k-1)+l(k-1)+1} \\
&\quad = -(-2)^{\mu_0} \sum_{l=j\,\mathrm{div}(k-1)+1}^{j\,\mathrm{div}(k-1)+\mu} (-\tfrac{1}{2})^{j\,\mathrm{div}(k-1)-l+1} e_{j\,\mathrm{mod}(k-1)+l(k-1)+1} \\
&\quad = -(-2)^{\mu_0} \sum_{l=1}^{\mu} (-2)^{l-1} e_{j+l(k-1)+1}
\end{aligned}
$$

for all j. Therefore, using (15), it follows that

$$
\begin{aligned}
\mathcal{E}_{i_0} - \mathcal{E}_{i_1} &= -(-2)^{\mu_1} E_{i_1} + (-2)^{\mu_0} E_{i_0} + 2^k((-2)^{\mu_1} E_{i_1-k} - (-2)^{\mu_0} E_{i_0-k}) \\
&\quad + 2^{k+1}((-2)^{\mu_1} E_{i_1-1} - (-2)^{\mu_0} E_{i_0-1}) \\
&\quad + \sum_{j=0}^{k-1} A_{i_0}(j)((-2)^{\mu_1} E_{i_1-k+j} - (-2)^{\mu_0} E_{i_0-k+j}) \\
&= -(-2)^{\mu_0} \sum_{l=1}^{\mu} (-2)^{l-1} \Bigg(e_{i_0+l(k-1)+1} - 2^k e_{i_0+(l-1)(k-1)} - 2^{k+1} e_{i_0+l(k-1)} \\
&\qquad\qquad - \sum_{j=0}^{k-1} A_{i_0}(j) e_{i_0+(l-1)(k-1)+j} \Bigg) \\
&= (-2)^{\mu_0} 2^k ((-2)^{\mu} e_{i_1} - e_{i_0}) \\
&\quad + (-2)^{\mu_0} \sum_{l=1}^{\mu} (-2)^{l-1} \left(e_{i_0+l(k-1)+1} - \sum_{j=0}^{k-1} A_{i_0}(j) e_{i_0+j+(l-1)(k-1)} \right).
\end{aligned}
$$

Appendix C. The Expected Number of Signatures

The expectation $\mathrm{E}_N(N)$ from (12) can be computed as follows using (11):

$$
\begin{aligned}
\mathrm{E}_N(N) &= \sum_{N=1}^{\infty} \Pr\{N \geq N\} \\
&= 1 + \sum_{N=2}^{k!-1} \prod_{i=1}^{N-2} \left(1 - \frac{i}{k!}\right)^{k-1}
\end{aligned}
$$

$$
\begin{aligned}
&= 1 + \sum_{N=2}^{k!-1} e^{(k-1)\sum_{i=1}^{N-2} \ln(1-i/k!)} \\
&\le 1 + \sum_{N=2}^{k!-1} e^{(k-1)\sum_{i=1}^{N-2} -i/k!} \\
&< 2 + \int_0^{\infty} e^{-x^2/(2k!/(k-1))}\,dx \\
&= 2 + \sqrt{\frac{\pi k!}{2(k-1)}}.
\end{aligned}
$$

Appendix D. The Complexity of the Attack

To compute the expectation E_N(# candidates) from (13), first note that

$$\mathrm{E}_N\binom{N}{2} = \sum_{N=2}^{\infty}\binom{N}{2}\Pr\{N = N\} = \sum_{N=2}^{\infty}(N-1)\Pr\{N \ge N\},$$

which follows from $\Pr\{N = N\} = \Pr\{N \ge N\} - \Pr\{N \ge N+1\}$ and

$$\binom{N}{2} = \binom{N-1}{2} + (N-1).$$

Then, using (11), computations similar to those above yield

$$
\begin{aligned}
\mathrm{E}_N(\text{\# candidates}) &= \sum_{N=2}^{\infty}(N-1)\cdot\prod_{i=1}^{N-2}\left(1-\frac{i}{k!}\right)^{k-1} \\
&< \int_0^{\infty} x\, e^{-(x-1)^2/(2k!/(k-1))}dx \\
&< \int_{-1}^{\infty} x\, e^{-x^2/(2k!/(k-1))}dx + \int_{-1}^{\infty} e^{-x^2/(2k!/(k-1))}dx \\
&< 1 + \frac{k!}{k-1} + 1 + \sqrt{\frac{\pi k!}{2(k-1)}}.
\end{aligned}
$$

References

[1] E. F. Brickell and K. S. McCurley, An interactive identification scheme based on discrete logarithms and factoring, *Journal of Cryptology* **5**(1) (1992), 29–39.

[2] D. Chaum, J.-H. Evertse, and J. van de Graaf, An improved protocol for demonstration possession of discrete logarithms and some generalizations, *Advances in Cryptology—Proceedings of Eurocrypt '87* (D. Chaum and W. L. Price, eds.), Lecture Notes in Computer Science, vol. 304, Springer-Verlag, Berlin, 1988, pp. 127–141.

[3] W. Diffie and M. E. Hellman, New directions in cryptography, *IEEE Transactions on Information Theory* **22**(6) (1976), 644–654.

[4] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory* **31**(4) (1985), 469–472.

[5] U. Feige, A. Fiat, and A. Shamir, Zero knowledge proofs of identity, *Journal of Cryptology* **1**(1) (1988), 77–95.

[6] C. G. Günther, An identity-based key-exchange protocol, *Advances in Cryptology—Proceedings of Eurocrypt* '89 (J.-J. Quisquater and J. Vandewalle, eds.), Lecture Notes in Computer Science, vol. 434, Springer-Verlag, Berlin, 1990, pp. 29–37.

[7] National Institute of Technology and Standards, *Specifications for the Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication XX, U.S. Department of Commerce, February 1, 1993.

[8] J. J. Quisquater and L. S. Guillou, A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory, *Advances in Cryptology—Proceedings of Eurocrypt* '88 (C. G. Günther, ed.), Lecture Notes in Computer Science, vol. 330, Springer-Verlag, Berlin, 1988, pp. 123–128.

[9] P. de Rooij, On the security of the Schnorr scheme using preprocessing, *Advances in Cryptology—Proceedings of Eurocrypt* '91 (D. W. Davies, ed.), Lecture Notes in Computer Science, vol. 547, Springer-Verlag, Berlin, 1991, pp. 71–80.

[10] C. P. Schnorr, Efficient identification and signatures for smart cards, *Advances in Cryptology—Proceedings of Crypto* '89 (G. Brassard, ed.), Lecture Notes in Computer Science, vol. 435, Springer-Verlag, Berlin, 1990, pp. 239–251.

[11] C. P. Schnorr, Efficient signature generation by smart cards, *Journal of Cryptology* **4**(3) (1991), 161–174.