Evaluation Report on DSA

Serge Vaudenay

IPA Work Delivery

Abstract. Following the contract between IPA and SV signed on 18.10.2001, this report presents security evaluation on DSA:
- the scheme and primitive of DSA
- the random generator given in FIPS 186-2 Appendix 3

In order to define the notations, we first summarize the DSA as presented in ANSI X9.30 Part 1 [1] and FIPS 186 [3].

Public parameters: integers p, q, g
p is a prime of L bits (L is at least 512, at most 1024, and a multiple of 64)
q is a prime of 160 bits and a factor of $p-1$
g is in $[1, p-1]$ and of order q modulo p
Secret key: integer x in $[1, q-1]$
Public key: $y = g^x \bmod p$
Signature generation for M: generate $k \in [1, q-1]$ and compute

$$r = (g^k \bmod p) \bmod q$$
$$s = \frac{\text{SHA-1}(M) + xr}{k} \bmod q$$

the signature is (r, s)
Signature (r', s') verification for M: check that r' and s' are in $[1, q-1]$ and that

$$r' = (g^{\frac{\text{SHA-1}(M)}{s'}} y^{\frac{r'}{s'}} \bmod p) \bmod q$$

SHA-1 is not specified FIPS 186 [3]. It is standardized in FIPS 180-1 [2] and the Part 2 of ANSI X9.30. The Appendices of ANSI X9.30 [1] and FIPS 186 [3] however specify how public parameters, secret keys and k values shall be generated. This will be discussed in the following sections.

1 Background

DSA comes from a long history of digital signature schemes. First, ElGamal has been studying applications of the discrete logarithm for cryptography in his PhD in 1984 [14–16]. The ElGamal signature scheme launched a dynasty of signature schemes based on the discrete logarithm problem. The original ElGamal problem suffers from several problems.

- efficiency: the signature is quite long (e.g. 2048 bits)
- security: some reasonable (but unfortunate) choices of public parameters lead to an attack due to Bleichenbacher [5].

The security was however proven in an ideal model for random choices of the public parameters by Pointcheval and Stern [19, 20]. Schnorr improved the algorithm by proposing a new signature scheme in 1989 [29, 30]. This was later adapted by NIST by submitting DSA in 1994.

2 Necessary Security Assumptions

Obviously, **ability to compute discrete logarithms** in the subgroup of $\mathbf{Z}_p^*$ spanned by g is a potential attack against DSA since we can recover x. A simple attack which uses Shanks baby step giant step algorithm works within **a complexity of** $\sqrt{q}$ which is about 2^{80}.

Inverting SHA-1 is another potential attack. If we can do so, one can first pick u_1 and u_2, compute $r = (g^{u_1}y^{u_2} \bmod p) \bmod q$, take $s = r/u_2 \bmod q$, compute $h = u_1 s \bmod q$ and invert SHA-1 on h in order to get M. (r, s) is then a valid signature for M. Brute force attack for inverting SHA-1 has **a complexity of** 2^{160}. **This can be decreased down to** 2^{80} **by precomputation**: instead of inverting on one single h, we precompute 2^{80} many (h, r, s) triplets, and we hash random messages until we find one of the precomputed h values.

Finding collisions on SHA-1 is another threat. If we know different M_1 and M_2 such that the corresponding message digests are the same, we can ask the legitimate signer for a signature on M_1, and we get a signature for M_2. The legitimate signer can also repudiate his signature for M_1 and claim that he signed M_2... The birthday paradox provides us with an attack of **complexity** 2^{80}.

Predicting the k issued by a pseudorandom generator is another attack. If one can guess k for an (r, s) signature, then it is fairly easy to recover x. In DSA the pseudorandom generator uses a seed of size b bits with $160 \leq b \leq 512$. So guessing the seed by brute force attack has **a complexity of** 2^b.

Therefore, we take as necessary assumptions that

- one cannot compute discrete logarithms within less than 2^{80} computations,
- one cannot invert SHA-1 within less than 2^{160} computations,
- one cannot forge a collision without doing more than 2^{80} computations,
- one cannot predict the output from the pseudorandom generator.

We can still wonder if these conditions are sufficient for the security and if they are satisfied.

3 SHA-1

SHA-1 is the last algorithm of series of hash functions: MD4, MD5, SHA, SHA-1. They are all based on the Merkle-Damgård paradigm [11, 18]: there is a compression function, and the message digest is computed by processing sequentially all blocks through the compression function. The Damgård result claims that if the compression function is collision resistant, then the overall construction is collision resistant as well. The compression function is made following the Davies-Meyer [12, 17] construction: each block is used as a key for an encryption function and there is a feed forward of the plaintext to the ciphertext in order to make it one-way.

MD4 [22–24] and MD5 [25] were designed by Rivest in 1990 and 1992 respectively. In 1991, Den Boer and Bosselaers [6] have found an attack against the last two round of MD4 (out of three). In 1995, Vaudenay [31] has found an attack against the first two rounds and a collision on all but a few bits. The year after, Dobbertin [13] came up with a complete collision on MD4.

MD5 has an additional round, but also a strange modification of the internal structure. This enabled Den Boer and Bosselaers [7] to forge a collision on the compression function of

MD5. Although this leads to no collision on MD5 itself, it invalidates the use of the Damgård theorem.

The internal structure of SHA is closer to MD4 than to MD5, but it hashes onto 160 bits instead of 128. It also has a kind of key schedule in order to transform the processed block from one round to the other. The key schedule is however linear and parallel: the 512 bits of each block are processed as 32 parallel sequences of 16 bits. This enabled Chabaud and Joux [10] to publish an attack against SHA in 1998. The attack would enable the forgery of a collision on SHA, but requires a complexity of 2^{61} computations (which was not implemented so far).

SHA-1 is an update of SHA which was proposed before the attack of Chabaud and Joux was released without any rationale. It is believed that the NSA has found the very same attack and asked to update the standard accordingly.

Since then **no attack was found against SHA-1** and it is believed to be a secure collision resistant one-way hash function.

4 Generation of Public Parameters

Although it suggested an optional algorithm for generating p, q, and g, the first versions of FIPS 186 [3] did not clearly specify how alternatives could be adopted.

As was shown by Vaudenay [32], authentication of these parameters must be mandatory. Actually, a better necessary condition for the security of DSA is that SHA-1 mod$q$ is collision-resistant. But, one can choose q so that it is not, even if SHA-1 is collision-resistant. This way, one can forge parameters for which DSA is insecure.

Similarly, no guaranty is given on g yet.

FIPS 186 [3] now explicitly says that **p, q and g must be given to the verifier in an authenticated manner**. Appendices propose a deterministic pseudorandom generator for making the p and q parameters, the proposed algorithm for g generation is optional. The seed is used in order to certify that they have been properly generated. In particular, they are not suspected to hide a trapdoor.

Note that the lack of certification for g can still induce a potential threat since the signer or verifier are not requested to check if the order of g is q and if this is "the right g". We can imagine a few possible attacks.

1. The verifier is given $g = 0$. This way we can easily forge signatures which will be accepted as valid by him. (Just take $r = 0$.)
2. The signer is given $g = 0$. This way released signatures lead to k and we can try to attack the pseudorandom generator.
3. The verifier is given $g = y^\alpha \bmod p$ with a known α. This way an attacker can forge a signature with public key y for any message digest h: he takes a random k, compute $r = (g^k \bmod p) \bmod q$, and $s = (h + r\alpha^{-1})/k \bmod q$.

We thus recommend to **generate g from the seed** as well.

Finally, we notice that the restriction on authenticated communication of the parameters implies that p, q, and g must be securely stored in memory, namely tamper proof against viruses, worms, ...

5 Bleichenbacher's Attack

FIPS 186 [3] specifies a way to generate the one-time key k in a pseudorandom way. Namely, it constructs a pseudorandom generator who generates a 160-bit number rand and takes $k = \text{rand} \mod q$. The output rand (which is based on SHA-1 or DES) is assumed to be uniformly distributed in $[0, 2^{160} - 1]$.

Quite recently (in February 2001), Bleichenbacher announced an impressive attack against DSA based on the fact that $\text{rand} \mod q$ is not uniformly distributed in $[1, q-1]$. More precisely any k in $[2^{160} - q + 1, q - 1]$ has a probability of 2^{-160} and any k in $[0, 2^{160} - q]$ has a probability of 2.2^{-160}. The main idea of this attack is to guess estimates $\tilde{x}$ of x and make a statistical analysis of $(\text{SHA-1}(M) + r\tilde{x})s^{-1} \bmod q$. When the estimate is correct, a bias is observe. The estimate is more and more precise until it yields the secret key x. Bleichenbacher estimates that the attack would be practical for a non negligible fraction of qs with a time complexity 2^{63}, a space complexity 2^{40}, and a collection of 2^{22} signatures. We believe that the attack can still be made more efficient.

The attack was impressive enough for NIST to **release a special publication in order to recommend a new pseudorandom generator [4]**: instead of taking $k = \text{rand} \mod q$, they recommend to generate two numbers rand and rand$'$ and to take $(\text{rand}||\text{rand}') \bmod q$. Let $n = 2^{320}$. Here any k in $[(n \bmod q) + 1, q - 1]$ has a probability of $\lfloor \frac{n}{q} \rfloor n^{-1}$ and any k in $[0, n \bmod q]$ has a probability of $(\lfloor \frac{n}{q} \rfloor + 1)n^{-1}$. Both probabilities are between $q^{-1} - n^{-1}$ and $q^{-1} + n^{-1}$.

Bleichenbacher's attack uses a bias of k defined by

$$\text{bias}(k) = E\left(e^{\frac{2i\pi k}{q}}\right).$$

When $k = \text{rand} \mod q$ with rand uniformly distributed in $[0, n-1]$, we have

$$\begin{aligned}
\text{bias}(k) &= \sum_{r=0}^{n-1} \frac{1}{n} e^{\frac{2i\pi(r \bmod q)}{q}} \\
&= \sum_{r=0}^{n-1} \frac{1}{n} e^{\frac{2i\pi r}{q}} \\
&= \frac{1}{n} \times \frac{e^{\frac{2i\pi n}{q}} - 1}{e^{\frac{2i\pi}{q}} - 1} \\
&= \frac{e^{i\pi\frac{n-1}{q}}}{n} \times \frac{\sin(\frac{\pi n}{q})}{\sin(\frac{\pi}{q})} \\
&\approx \frac{q e^{i\pi\frac{n-1}{q}}}{\pi n} \times \sin(\frac{\pi n}{q})
\end{aligned}$$

Obviously, when n is close to q, say $q = n - \varepsilon$, we have $\text{bias}(k) \approx -\frac{\varepsilon}{n} e^{i\pi\frac{n-1}{q}}$ which is small. For primes q arbitrarily chosen of size 160 and $n = 2^{160}$, the sinus can be quite large and the norm of the bias can be within the order of magnitude of $\frac{1}{\pi}$. When n is within the order of magnitude of q^2 (as in the fix recommended by NIST), the norm is less than q^{-1}.

Our comment is that both Bleichenbacher's attack and NIST's recommendation are still underground research: none have been published in the academic world yet, and the impact and limitations of the cryptanalytic branch opened by Bleichenbacher are not yet quite clear. Although the NIST fix looks quite reasonable, wisdom recommends to **let the pseudorandom generator open** in the standard for later release in case of new research results.

6 Pseudorandom Generator for Secret Keys and *k* Values

FIPS 186 [3] suggests that random numbers must be generated according to a deterministic pseudorandom generator. These values include the secret keys x and the one-time keys k. FIPS 186 [3] and the fix to Bleichenbacher attack [4] can be described as follows.

1. We construct a primitive G who takes two inputs: a 160-bit parameter t (the *type*) which is a constant specific to the number we want to generate (x or k), and a b-bit ($160 \leq b \leq 512$) key which is also specific to the number we want to generate. The primitive outputs a 160-bit number. Two constructions for G are proposed.
   The first one is based on the compression function of SHA-1. This means that we do not consider message padding, iteration, and initialization as in the Merkle-Damgård [11, 18] construction. The registers of the compression function are initialized with the type value. The message block consists of the key padded with zero bits up to 512 bits. The output of the compression function (obtained after the feed forward due to the Davies-Meyer scheme [12, 17]) is the output of G.
   The second one is based on DES. It consists of the following steps.
   $$\begin{aligned} t &= t_1||t_2||t_3||t_4||t_5 \\ c &= c_1||c_2||c_3||c_4||c_5 \\ x_i &= t_i \oplus c_i \\ y_{i,1}||y_{1,2} &= \mathrm{DES}_{c_{i+4}||c_{i+3}}(x_i||(x_{i+1} \oplus x_{i+4})) \\ z_i &= y_{i,1} \oplus y_{i+2,2} \oplus y_{i+3,1} \\ G(t,c) &= z_1||z_2||z_3||z_4||z_5 \end{aligned}$$
   with $i = 1, 2, 3, 4, 5$ where subscripts are considered modulo 5.
2. We construct from G a pseudorandom generator which generates 160-bit numbers which is specific to a given type t and we denote rand_t. The generator can use a user defined input which is called seed, and uses a seed which is surprisingly called key_t. It works as follows.
   (a) $\mathrm{rand}_t \leftarrow G(t, \mathrm{key}_t + \mathrm{seed} \bmod 2^b)$
   (b) $\mathrm{key}_t \leftarrow 1 + \mathrm{key}_t + \mathrm{rand}_t \bmod 2^b$
3. We construct a specific pseudorandom generator. For FIPS 186 [3], it simply consists in taking $\mathrm{rand}_t \bmod q$. For [4], it consists in calling the pseudorandom twice in order to get rand_t and rand'_t and taking $(\mathrm{rand}_t||\mathrm{rand}'_t) \bmod q$.

Note that **the system is vulnerable against many kinds of replay attacks**. If the signer is a tamper proof device, assuming that we can clone it, then we can ask the two clones to sign different messages. We will then get two s signatures with the same unknown k and

x and for different messages. It is then easy to recover x. If the signer's seed is in a hard disk (encrypted or not), it may be feasible to restore the hard disk in one of its previous state (either by the attack or by a restore from backup due to computer crash). We can also get two signatures which use the same k values this way. **Implementation therefore needs extra care.**

Considering the pseudorandom generator as a random function with an internal state of b bits, we know that the expected loop length is within the order of magnitude of $\sqrt{2^b}$. For $b = 160$, we start having **security problems after about 2^{80} signatures**.

For the specific pseudorandom generators proposed in FIPS 186 [3], we observed that **the DES-based one has strange properties**. For instance, if all t_i are equal, as well as all c_i, then the generated $G(t, c)$ has the same property. More interestingly, the cyclic shift by 32 bits R has the property that $G(R(t), R(c)) = R(G(t, c))$. In particular, since the t value of the generator for k is obtained by R on the t value for the generator for x, if both seeds are related by R, the two generators generate x and k such that $k = R(x)$... Although this does not seem to be dangerous for DSA, it is obviously the kind of remarkable property that a pseudorandom generator should not have. Therefore we think that **the DES-based pseudorandom generator is not mature enough** in the academic sense.

For completeness, we mention that the precomputation of (k, r) pairs with a few computations is an old problem. It was first considered by Schnorr in his submission of the Schnorr signature scheme [29, 30]. The two consecutive precomputation algorithm versions proposed in 1989 and 1991 have successively been broken by de Rooij [26–28].

7 Security of DSA in an Ideal Model

Two different security results in ideal models are available from Pointcheval and Vaudenay [21], and Brickell et al. [8]. In both, we consider SHA-1 as a random oracle H. In the first one, we also consider $r = H_1(g^k \bmod p)$ with a random oracle H_1 instead of $r = g^k \bmod p \bmod q$. Hypothesis are

- *the random oracles H and H_1 are uniformly distributed*
- *the output of H and H_1 are subsets of* $[0, q - 1]$.

The first step of the security result is as follows.

> *An adaptive attack which can output an existential signature forgery can be transformed into an adaptive attack which extract the secret key x.*

The result is also quantitative.[1] Note that the result is quite idealistic. In particular, the hypothesis on H_1 (the modulo q reduction) as a random oracle is **not very convincing**. Furthermore, this step "only" proves that signature forgery is equivalent to secret key awareness. But there in no clue about information leakage from the signature. Namely, it does not prove that finding the secret key with an adaptive attack is hard. This is actually not the case without any additional assumption on the randomness of k as demonstrated by Bleichenbacher.

Nevertheless, a second step can be performed with the additional assumption

[1] See [21] or [8] for more information.

– *all k values are independently and uniformly generated in $\mathbf{Z}_q^*$ (or indistinguishable from such a generator).*

With this we prove that we can simulate the signature oracle, which proves the following result.

> *An adaptive attack which can output an existential signature forgery can be transformed into an algorithm which computes the secret key x.*

Here, access to the signing oracle is removed. Therefore, **the signature is provably secure provided that H and H_1 can be approximated by random oracles**, that the generator for k is secure, and that the discrete logarithm problem is hard.

In the second result, we assume that the modulo q mapping from the subgroup of $\mathbf{Z}_p^*$ spanned by g to $[0, q-1]$ has no ℓ-collision (namely ℓ pairwise different inputs which produce the same output) for $\ell = O(\log q)$, and that $H(M)$ is replaced by $H(r, M)$ as in the Schnorr signature [29, 30]. (So this is not the standard scheme.) Hypothesis and security results are similar. Although the hypothesis on the modulo q reduction is reasonable, the signature scheme has been modified so the result is not applicable for DSA.

Brown [9] also proved the security in another model. (This was done for ECDSA, but it holds for DSA as well.) In his model, **group operations are performed by a random oracle**, but H_1 and SHA-1 are fixed. He also assumes that H_1 is invertible (which is the case, since it is the modulo q reduction). Brown basically proves that **signature forgery is equivalent to inverting SHA-1 for passive attacks** (i.e. no-message attacks) and **to finding collisions for active attacks**. (Provided that pseudorandom generators are good!) However, the model is not very realistic for DSA since we know better algorithms than generic ones in $\mathbf{Z}_p^*$.

8 Conclusion

We have shown that DSA is vulnerable against attacks of complexity 2^{80}. The pseudorandom generator for keys and one-time keys has just been changed for security reasons. Since this has not been addressed by the academic community at the time this report was written, it would be wise to prepare potential modifications of this part in the future.

We pointed out an unexpected property of the DES-based G generator. Therefore we think that this part of the standard must be reconsidered. In the meantime we recommend to use the SHA-based generator.

We also outline that implementations may easily lead to security issues by kinds of replay attacks due to the deterministic pseudorandom generator. This would deserve informative security warnings in the standard.

DSA is provably secure in the random oracle model, provided

1. that the generator for k is secure,
2. that the SHA-1 hash function can be approximated by a random oracle,
3. that the discrete logarithm is hard in the corresponding group,
4. and that we approximate the mapping from the exponential of k to r by a random oracle.

The last assumption does not make reasonable sense in case of DSA though. The first assumption was wrong as shown by Bleichenbacher. The second and last assumptions are however quite artificial.

DSA is provably secure in the generic group model, provided

1. that the generator for k is secure,
2. and that the SHA-1 hash function is collision-resistant.

(We know that the discrete logarithm is hard in this model, so there is no reason to assume it further.) The generic group assumption is however not valid in the case of DSA.

DSA faces to the problem of parameters validation: parameters need to be authentic and trapdoor-free. For this a pseudorandom generation algorithm needs to be specified. This was half done for DSA since g is still missing. Hence we recommend to update the public parameter selection algorithms: DSA should generate g in a verifiable way as well.

References

1. ANSI X9.30. Public Key Cryptography for the Financial Services Industry: Part 1: The Digital Signature Algorithm (DSA). American National Standard Institute. American Bankers Association. 1997.
2. Secure Hash Standard. *Federal Information Processing Standard* publication #180-1. U.S. Department of Commerce, National Institute of Standards and Technology, 1995.
3. Digital Signature Standard (DSS). *Federal Information Processing Standards* publication #186-2. U.S. Department of Commerce, National Institute of Standards and Technology, 2000.
4. Recommendations Regarding Federal Information Processing Standard (FIPS) 186–2, Digital Signature Standard (DSS). NIST Special Publication 800–XX. U.S. Department of Commerce, National Institute of Standards and Technology, October 2001.
5. D. Bleichenbacher. Generating ElGamal Signatures without Knowing the Secret Key. In *Advances in Cryptology EUROCRYPT'96*, Zaragoza, Spain, Lectures Notes in Computer Science 1070, pp. 10–18, Springer-Verlag, 1996.
6. B. den Boer, A. Bosselaers. An Attack on the Last two Rounds of MD4. In *Advances in Cryptology CRYPTO'91*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 576, pp. 194–203, Springer-Verlag, 1992.
7. B. den Boer, A. Bosselaers. Collisions for the Compression Function of MD5. In *Advances in Cryptology EUROCRYPT'93*, Lofthus, Norway, Lectures Notes in Computer Science 765, pp. 293–304, Springer-Verlag, 1994.
8. E. Brickell, D. Pointcheval, S. Vaudenay, M. Yung. Design Validations for Discrete Logarithm Based Signature Schemes. In *Public Key Cryptography*, Melbourne, Australia, Lectures Notes in Computer Science 1751, pp. 276–292, Springer-Verlag, 2000.
9. D.R.L. Brown. The Exact Security of ECDSA. Technical Report CORR 2000–34, Certicom Research, 2000. `http://www.cacr.math.uwaterloo.ca`
10. F. Chabaud, A. Joux. Differential Collisions in SHA-0. In *Advances in Cryptology CRYPTO'98*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 1462, pp. 56–71, Springer-Verlag, 1998.
11. I. B. Damgård. A Design Principle for Hash Functions. In *Advances in Cryptology CRYPTO'89*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 435, pp. 416–427, Springer-Verlag, 1990.
12. R. W. Davies, W. L. Price. Digital Signature – an Update. In *Proceedings of the International Conference on Computer Communications*,Sydney, pp. 843-847, North-Holland, 1985.
13. H. Dobbertin. Cryptanalysis of MD4. In *Fast Software Encryption*, Cambridge, United Kingdom, Lectures Notes in Computer Science 1039, pp. 53–69, Springer-Verlag, 1996.
14. T. ElGamal. Cryptography and Logarithms over Finite Fields. PhD Thesis, Stanford University, 1984.
15. T. ElGamal. A Public-key Cryptosystem and a Signature Scheme based on Discrete Logarithms. In *Advances in Cryptology CRYPTO'84*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 196, pp. 10–18, Springer-Verlag, 1985.
16. T. ElGamal. A Public-key Cryptosystem and a Signature Scheme based on Discrete Logarithms. *IEEE Transactions on Information Theory*, vol. IT-31, pp. 469–472, 1985.
17. S. M. Matyas, C. H. Meyer, J. Oseas. Generating Strong One-way Functions with Cryptographic Algorithm. *IBM Technical Disclosure Bulletin*, vol. 27, pp. 5658–5659, 1985.
18. R. C. Merkle. One way Hash Functions and DES. In *Advances in Cryptology CRYPTO'89*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 435, pp. 416–427, Springer-Verlag, 1990.

19. D. Pointcheval, J. Stern. Security Proofs for Signature Schemes. In *Advances in Cryptology EUROCRYPT'96*, Zaragoza, Spain, Lectures Notes in Computer Science 1070, pp. 387–398, Springer-Verlag, 1996.
20. D. Pointcheval, J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, vol. 13, pp. 361–396, 2000.
21. D. Pointcheval, S. Vaudenay. On Provable Security for Digital Signature Algorithms. Technical report LIENS 96-17, Ecole Normale Supérieure, 1996.
22. R. L. Rivest. The MD4 Message Digest Algorithm. In *Advances in Cryptology CRYPTO'90*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 537, pp. 303–311, Springer-Verlag, 1991.
23. R. L. Rivest. The MD4 Message Digest Algorithm. RFC 1186, Oct 1990.
24. R. L. Rivest. The MD4 Message Digest Algorithm. RFC 1320, Apr 1992.
25. R. L. Rivest. The MD5 Message Digest Algorithm. RFC 1321, Apr 1992.
26. P. de Rooij. On the Security of the Schnorr Scheme Using Preprocessing. In *Advances in Cryptology EUROCRYPT'91*, Brighton, United Kingdom, Lectures Notes in Computer Science 547, pp. 71–80, Springer-Verlag, 1991.
27. P. de Rooij. On Schnorr's Preprocessing for Digital Signature Schemes. In *Advances in Cryptology EUROCRYPT'93*, Lofthus, Norway, Lectures Notes in Computer Science 765, pp. 435–439, Springer-Verlag, 1994.
28. P. de Rooij. On Schnorr's Preprocessing for Digital Signature Schemes. *Journal of Cryptology*, vol. 10, pp. 1–16, 1997.
29. C. P. Schnorr. Efficient Identification and Signature for Smart Cards. In *Advances in Cryptology CRYPTO'89*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 435, pp. 235–251, Springer-Verlag, 1990.
30. C. P. Schnorr. Efficient Identification and Signature for Smart Cards. *Journal of Cryptology*, vol. 4, pp. 161–174, 1991.
31. S. Vaudenay. On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER. In *Fast Software Encryption*, Leuven, Belgium, Lectures Notes in Computer Science 1008, pp. 286–297, Springer-Verlag, 1995.
32. S. Vaudenay. Hidden Collisions on DSS. In *Advances in Cryptology CRYPTO'96*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 1109, pp. 83–88, Springer-Verlag, 1996.