ECOLE NORMALE SUPERIEURE

Hidden Collisions on DSS

Serge VAUDENAY

LIENS - 96 - 9

Département de Mathématiques et Informatique

CNRS URA 1327

Hidden Collisions on DSS

Serge VAUDENAY

LIENS - 96 - 9

June 1996

Laboratoire d'Informatique de l'Ecole Normale Supérieure
45 rue d'Ulm 75230 PARIS Cedex 05

Tel : (33)(1) 44 32 00 00

Adresse électronique : ... @dmi.ens.fr

Hidden Collisions on DSS

Serge Vaudenay*

Ecole Normale Supérieure — DMI
45, rue d'Ulm
75230 Paris Cedex 5 France
`Serge.Vaudenay@ens.fr`

Abstract

We explain how to forge public parameters for the Digital Signature Standard with two known messages which always produce the same set of valid signatures (what we call a collision). This attack is thwarted by using the generation algorithm suggested in the specifications of the Standard, so it proves one always need to check proper generation. We also present a similar attack when using this generation algorithm within a complexity 2^{74}, which is better than the birthday attack which seeks for collisions on the underlying hash function.

Imagine you want to join to a brand new association which offers to provide useful services on the net. To allow electronic payment, this association provides a DSS implementation with public parameters

$$\begin{aligned} p \;=\; & 1007386175274283816733054843443587432664299802160928 \\ & 9724334546391745980774853739798193524368725087200030 \\ & 875184211970398850090583601122813103828861440790761 \end{aligned}$$

and

$$q = 759902064211816970120975637406935605590678547999.$$

To check the connection, the server requests you to sign the message "`This is just a test`". Then time goes, and your bank warns you that you need to feed your account after your last check of $\$9,302$. Of course, the manager of the association has just disappeared with the money obtained from all the 215 members! (This is a $\$2,000,000$ swindle.)

**Laboratoire d'Informatique de l'Ecole Normale Supérieure*, research group affiliated with the CNRS

This attack comes from a special forgery of the public parameter. It is *not* applicable to the accurate DSS suggested in [2] since it requires the production of a certificate of *good* forgery. This attacks however proves that the signer must check the certificate himself (which is not explicitly mentioned is the Standard), which may be a very cumbersome task for low cost devices.

In this paper, we explain how to perform this attack. We also show a similar attack against DSS with the suggested certificate of good forgery with a complexity equivalent to 2^{74} SHS computations. The complexities of the attacks do not depend on the length of the prime p used in the signature scheme.

1 Signatures based on discrete logarithm

The first digital signature algorithm based on the discrete logarithm problem (namely, the discrete log problem in the group $\mathbb{Z}_p{}^*$, given a prime number p) was the ElGamal signature [4]. This scheme produces quite long signatures and is also subject to Bleichenbacher's attack which uses small factors of $p-1$ [3].

The Schnorr signature [6, 7] repars those two shortcomings by using an element g whose order is a 160-bit prime factor q of $p-1$. The underlying group for the discrete log problem is thus a subgroup of $\mathbb{Z}_p{}^*$ with order q. The Digital Signature Standard (DSS) [2] also uses such a g.

In the following, we only consider the case of the DSS scheme. In the signature scheme, the message is processed through the Secure Hash Standard (SHS) [1] which produces a 160-bit digest. This value then appears as a power of g. Hence, the *real* hash value is not the output of SHS, but rather the message digest reduced modulo q. Since q is less than the largest output of SHS, this may produce collisions.

For completeness, we now recall the outlines of DSS.

p, q and g are the public parameters chosen by the authority. p is a 512-bit prime (or a 1024-bit prime in stronger versions), q is a 160-bit prime factor of $p-1$, and g is a primitive qth root of 1 modulo p. Each user has a secret key x (which is a 160-bit integer) and publishes a corresponding 512-bit public key $y = g^x \bmod p$. The signature of an arbitrary message m using a (secret) fresh random 160-bit integer k is a (r, s) pair of 160-bit integers defined by

$$\begin{aligned} r &= (g^k \bmod p) \bmod q \\ s &= \frac{\mathrm{SHS}(m) + xr}{k} \bmod q. \end{aligned}$$

The verification of the signature is performed by checking

$$r = \left(g^{\frac{\mathrm{SHS}(m)}{s} \bmod q} y^{\frac{r}{s} \bmod q} \bmod p \right) \bmod q.$$

2 Collision for DSS

We have noticed that the *real* hash function which is used in DSS is SHS mod q. Hence, if we know a pair of messages (m, m') such that

$$\mathrm{SHS}(m) \equiv \mathrm{SHS}(m') \pmod q$$

any signature from any user of message m is also a valid signature of message m'. We call such a pair a *collision* for DSS.

Of course, since SHS is considered as a *secure* hash function, it is still infeasible to get a collision. More precisely, the complexity of the best attack (based on the birthday paradox) has a complexity of 2^{80}. Similarly, it may be infeasible, given a 160-bit prime q, to find a collision on SHS mod q. It is however possible to construct q from an (un)willing collision (m, m').

More concretely, from a random pair (m, m'), we can check whether or not $q = |\mathrm{SHS}(m) - \mathrm{SHS}(m')|$ is a 160-bit prime. The integer $|\mathrm{SHS}(m) - \mathrm{SHS}(m')|$ is obviously a 160-bit integer with probability 1/2. Then, a random 160-bit integer is a prime with probability approximately $1/160 \log 2 \approx 1/111$, thanks to the Prime Number Theorem. Hence, with an average of 222 trials, we obtain a collision which defines a valid prime q. It is not difficult, given a prime q, to issue valid p and g, for instance by following the generation algorithm provided in [2].

The example given in the introduction uses the message

$$m = \texttt{This is just a test}$$

which is encoded as

$$\begin{array}{llll}
54686973 & 20697320 & 6a757374 & 20612074 \\
65737480 & 00000000 & 00000000 & 00000000 \\
00000000 & 00000000 & 00000000 & 00000000 \\
00000000 & 00000000 & 00000000 & 00000098
\end{array}$$

in hexadecimal and its hash value is

$$\mathrm{SHS}(m) = 653095248274681173784642234520335115855431883588.$$

The second message is

$$m' = \texttt{Transfer \$9,302 on account XYZ}$$

and its hash value is

$$\mathrm{SHS}(m') = 1412997312486498143905617871927270721446110431587.$$

So let

$$q = \mathrm{SHS}(m) - \mathrm{SHS}(m')$$

which is a 160-bit prime. Then one can issue a 512-bit prime p such that q divides $p-1$.

3 The public parameters generation of DSS

The description of DSS suggests that q is first generated from a random seed by a specific algorithm, then p and g are issued from q by the same seed. The seed should be used as a certificate of a honest forgery for p and q. The previous attack suggests that this feature must be used.

Is the attack really thwarted by this generation algorithm? We recall that q is set to

$$(\mathrm{SHS}(\mathit{seed}) \oplus \mathrm{SHS}(\mathit{seed}+1)) \vee 2^{159} \vee 1$$

until it is a prime where $\oplus$ and $\vee$ denote the bitwise exclusive and inclusive or respectively. (We notice that we need in average about 55 trials to get a prime q since this is a random 160-bit odd integer.) Let $\mathit{seed} = m$ and $\mathit{seed}+1 = m'$. The same attack holds whenever

$$|\mathrm{SHS}(\mathit{seed}) - \mathrm{SHS}(\mathit{seed}+1)| = (\mathrm{SHS}(\mathit{seed}) \oplus \mathrm{SHS}(\mathit{seed}+1)) \vee 2^{159} \vee 1$$

Since this occurs with probability $\frac{1}{2} \times \frac{1}{4} \times \left(\frac{3}{4}\right)^{157} \approx 2^{-68.16}$, we obtain that $2^{73.95}$ trials are required on average to mount this attack.

The Standard says that parameters p and q shall be generated as suggested above, or using other FIPS approved security methods. This study may thus be helpful for proposing other generation algorithms. For instance, we can suggest to set

$$q = \mathrm{SHS}(seed) \vee 2^{159} \vee 1$$

until q is valid.

4 On the g parameter

Surprisingly, no particular care is required for parameter g. Thus, fake gs like $g = 0$ or 1 are implicitly accepted! Actually, a dishonest authority can provide $g = 1$ to a user and make him accept any signature forged by

$$\begin{aligned} r &= (y^k \bmod p) \bmod q \\ s &= \frac{r}{k} \bmod q. \end{aligned}$$

In a more dedicated attack, the authority provides $g = y^\alpha \bmod p$ to Alice, where y is the public key of Bob, and forges Bob's signatures by

$$\begin{aligned} r &= (y^k \bmod p) \bmod q \\ s &= \frac{\alpha \mathrm{SHS}(m) + r}{k} \bmod q. \end{aligned}$$

We thus suggest to add a certificate of valid forgery of g when issuing the public parameters.

5 Conclusion

We performed an attack against DSS which allows the issuing autority to forge valid public parameters with hidden collisions. It confirms internal problems in the signature scheme which were already predicted by Pointcheval and Stern in [5] because of not using a random pattern in the hash function like in the Schnorr signature [6]. We have proved that when the issuing authority is not trusted, all users must then check proper generation of the public parameters. We also showed how to adapt this attack to the generation algorithm suggested in the Standard within a complexity 2^{74}. Even if the complexity were smaller, the attack would still not be so dramatic because it is easy to detect. This should be registered as an existing (bad) property of DSS though.

This attack can easily be avoided by using a 161-bit prime q, or by dropping the most significant bit of SHS (or by setting the least significant bit to a constant before using it in the signature scheme), or by padding a random pattern before hashing. Its complexity can also be increased up to 2^{80} by using a stronger certificate-based generation algorithm for p and q.

We also presented attacks based on malicious forgery of the g parameter. We thus recommand to use a suitable certificate of honest forgery for g as well as for p and q.

References

[1] U. S. Department of Commerce, National Institute of Standards and Technology. Secure Hash Standard. Federal Information Processing Standard Publication 180-1, 1995.

[2] U.S. Department of Commerce, National Institute of Standards and Technology. Digital Signature Standard. Federal Information Processing Standard Publication 186, 1994.

[3] 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.

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

[5] 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.

[6] 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. 239–252, Springer-Verlag, 1990.

[7] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, vol. 4, pp. 161–174, 1991.