The Security of DSA and ECDSA

Bypassing the Standard Elliptic Curve Certification Scheme

Serge Vaudenay

Swiss Federal Institute of Technology (EPFL)
Serge.Vaudenay@epfl.ch

Abstract. DSA and ECDSA are well established standards for digital signature based on the discrete logarithm problem. In this paper we survey known properties, certification issues regarding the public parameters, and security proofs.
ECDSA also includes a standard certification scheme for elliptic curve which is assumed to guarantee that the elliptic curve was randomly selected, preventing from any potential malicious choice. In this paper we show how to bypass this scheme and certify any elliptic curve in characteristic two. The prime field case is also studied. Although this does not lead to any attack at this time since all possible malicious choices which are known at this time are specifically checked, this demonstrates that some part of the standard is not well designed. We finally propose a tweak.

DSA was published in 1994 following a long dynasty of digital signature schemes based on the ElGamal scheme [10,11,12]. Since then an extensive literature addressed security analysis, performances, and variants. Among the famous variants ECDSA was proposed in 1998. In this paper we aim to survey dedicated attacks and provable security. We also address the parameter validation issue. In particular we show that we may be able to maliciously choose an elliptic curve for ECDSA despite the standard validation scheme.

1 DSA and ECDSA

In order to define the notations, we first summarize the DSA as presented in ANSI X9.30 Part 1 [1] and FIPS 186 [5].

Public parameters: integers p, q, g and a seed in order to validate q
- 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

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

If $r = 0$ or $s = 0$, try again. The signature is (r, s)

Signature (M, r', s') verification: 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 in FIPS 186 [5]. It is standardized in FIPS 180-1 [4] and the Part 2 of ANSI X9.30. The Appendixes of ANSI X9.30 [1] and FIPS 186 [5] however specify how public parameters, secret keys and k values shall be generated. They do not specify how the parameters validity should be checked. They simply say that the parameters must be transmitted in an authenticated way.

Let us now summarize the ECDSA as presented in ANSI X9.62 [2].

Public parameters: finite field $\mathbf{F}_q$ and a field representation choice, two parameters a and b which define an elliptic curve C over $\mathbf{F}_q$, a seed which validates C, a prime integer $n > 2^{160}$, and a point $G \in C$ of order n. Here q is either prime or a power of 2

Secret key: integer d in $[1, n-1]$

Public key: $Q = dG$

Signature generation for M: generate $k \in [1, n-1]$ and compute

$$(x_1, y_1) = kG$$
$$r = \overline{x_1} \bmod n$$
$$s = \frac{\text{SHA-1}(M) + dr}{k} \bmod n$$

If $r = 0$ or $s = 0$, try again. The signature is (r, s)

Signature (M, r', s') verification: check that r' and s' are in $[1, n-1]$ and that $r' = \overline{x_1} \bmod n$ for $(x_1, y_1) = u_1 G + u_2 Q$, $u_1 = \frac{\text{SHA-1}(M)}{s'} \bmod n$, and $u_2 = \frac{r'}{s'} \bmod n$

Here $\overline{x_1}$ is simply a way of converting a field element into an integer and SHA-1 is a hash function specified in FIPS 180-1 [4].

The signature is a pair of integers. The public key is a point on a curve. So ANSI X9.62 [2] needs to define a standard way for representing an integer, a point, and therefore a field element. In addition we need a standard way to represent the public parameters: the field representation, the curve definition, ... ANSI X9.62 [2] extensively defines all this.

Additionally, users need to check if the public parameters are valid as follows.

1. Check that q is an odd prime or a power of 2. In the latter case, check that the field representation choice is valid.

2. Check that a, b, x_G, y_G where $G = (x_G, y_G)$ lies in $\mathbf{F}_q$.
3. Check that seed certifies a and b. (This point will be discussed in Section 5.)
4. For q prime, check that $4a^3 + 27b^2 \bmod q \neq 0$. For q a power of two, check that $b \neq 0$.
5. Check that G lies in C.
6. Check that n is a prime greater than 2^{160} and $4\sqrt{q}$.
7. Check that $nG = \mathcal{O}$, the neutral element in C.
8. Check the MOV and anomalous condition for C.

The verifier further validates the public key by checking that $Q \neq \mathcal{O}$, $Q \in C$, and $nQ = \mathcal{O}$.

2 Dedicated Attacks

In this section we survey some known properties of DSA and ECDSA.

2.1 Signature Manipulation in ECDSA

Interestingly, the $(x_1, y_1) \mapsto \overline{x_1} \bmod n$ function does not use the information about y_1. We have two points in the elliptic curve with the same x_1 coordinate which happen to be opposite of each other. (Hence dropping y_1 looses one bit of information.) It means that replacing k by $-k \bmod n$ would lead to the same x_1 hence the same r. This manipulation replaces s by $-s \bmod n$. Hence we can replace any (r, s) signature by $(r, -s \bmod n)$ which is another valid signature for the same message.

The drop of one bit has the other consequence that one can choose his secret key in order to create a valid signature for two different messages simultaneously as pointed out by Stern et al. [24]. Indeed we can just compute r from a random k then select

$$d = -\frac{\text{SHA-1}(M_1) + \text{SHA-1}(M_2)}{2r} \bmod n.$$

2.2 Bleichenbacher Attack against the Pseudorandom Generator

The initial standard pseudorandom generator in DSA for k was simply a 160-bit pseudorandom number reduced modulo q. Bleichenbacher[1] observed that the probability of k in the $[0, 2^{160} - q]$ range have probability which is twice of the others. This leads to a bias

$$E\left(e^{\frac{2i\pi k}{q}}\right) \approx \frac{q e^{i\pi\frac{N-1}{q}}}{\pi N} \times \sin\left(\frac{\pi N}{q}\right)$$

where $N = 2^{160}$. Since $q \approx N$, this may be large depending on the $\frac{\pi N}{q}$ angle. Bleichenbacher actually used it in order to approximate the secret key more and more precisely with signatures. Based on that the standard was tweaked by basically replacing N by N^2. (See [6].)

The same remark holds for ECDSA with n instead of q (but the $\frac{\pi N}{n}$ angle is very small most of the time).

[1] Private communication.

2.3 Restart Attack

Assuming that the pseudorandom generator for k is deterministic and that one can reset the internal state of the generator, then we can break the scheme with signatures of two different messages: if the signer signs M_1 by generating k and we can reset it so that it generates the same k for M_2, we have a signature (r, s_1) for M_1 and a signature (r, s_2) for M_2. Hence we obtain that

$$x = -\frac{s_2\text{SHA-1}(M_1) - s_1\text{SHA-1}(M_2)}{r(s_2 - s_1)} \bmod q.$$

This attack model makes sense if we have a clone of the signer with the same initial state.

A similar attack holds for ECDSA.

3 Parameter Validation

In this section we survey some parameter certification issues.

3.1 Public Keys Certificate

Authentication of public keys is a well known problem. It can be solved by using certificates which are basically signatures of the public key by a certificate authority. Certificates then rely on the authentication of the certificate authority public key. This is still an important issue since there is no other mean than physical protection: when delivered, the public key needs to be manually authenticated, then physically protected in the memory.

We show in the next sections that we have similar issues for the public parameters.

3.2 p and q Validation

As pointed out by Vaudenay [25], one can choose p and q in DSA such that a collision on SHA-1 mod q is known. One simply take random $q = \text{SHA-1}(M_1) - \text{SHA-1}(M_2)$ until it is a 160-bit prime number and take random $p = aq + 1$ until it is a prime. With this choice one can forge a signature for M_2 with a signature of M_1.

In order to avoid this attack we generate p and q following a standard generator and use the initial seed as a certificate of good forgery. As specified in [5], q is generated by

$$q = (\text{SHA-1}(\text{seed}) \oplus \text{SHA-1}(\text{seed} + 1)) \vee 2^{159} \vee 1$$

until it is valid where $\oplus$ and $\vee$ denote the bitwise XOR and OR operations. This means that we take the XOR of two random values coming out from SHA-1 and we force the least and most significant bits to 1. The certificate for p and q is thus simply the seed.

As pointed out in [25], the attack still holds whenever

$$\text{SHA-1}(\text{seed}) = \text{SHA-1}(\text{seed} + 1) \pm q$$

which occurs with probability $2 \times \frac{1}{4} \times \left(\frac{3}{4}\right)^{158} \times \frac{1}{4} \approx 2^{-68.6}$. (2 is for $\pm$, each $\frac{1}{4}$ is for a difference equal to 1 without carry bit, each $\frac{3}{4}$ is for a difference without carry bit).[2] Therefore it takes roughly 2^{80} trials in order to get a seed which satisfies the condition due to the additional overhead due to the primality tests. This is within the order of magnitude of the brute force attacks which are discussed in Sec. 4.1.

3.3 g Validation

As pointed out in [25], there is no similar certificate for g (resp. G). If we had no verification on g at all, we may have attacks against a given DSA signature verifier as follows. (Similar attacks hold for ECDSA.)

Replacing g by 0. If we can corrupt g in the memory of the verifier we can replace it by 0. Then any signature with $r = 0$ becomes valid for any public key.

Replacing g by 1. If we can corrupt g in the memory of the verifier we can replace it by 1. Then we can forge a signature for any message for a given public key y by picking random $r = (y^\alpha \bmod p) \bmod q$, then $s = \frac{r}{\alpha} \bmod q$.

Other replacement. One may want to check that g has an order of q which would thwart the last two attacks. However we can still replace g by a random power of y. In this case the attacker knows the discrete logarithm of y in this basis and can sign any message.

4 Provable Security

In this section we survey provable security results for DSA and ECDSA.

4.1 Necessary Conditions

Theorem 1. *Here are necessary security conditions for DSA (resp. ECDSA).*

1. *The discrete logarithm in the subgroup spanned by g (resp. G) is hard.*
2. *SHA-1 is a one-way hash function.*
3. *SHA-1 is a collision-resistant hash function.*
4. *The generator for k is unpredictable.*

As will be noticed later these conditions are “more or less” sufficient in some particular models.

[2] In [25] a probability of $2^{-68.16}$ was given but there was a computation error in the estimate.

Proof. Condition 1 is quite obvious: if the condition does not hold, we can just compute the discrete logarithm of the public key and obtain the secret key. (One need to randomize the public key by standard whitening techniques.) One should however notice that the discrete logarithm problem is not equivalent to computing the secret key. The legitimate signer indeed knows how to compute it but does not necessarily know how to solve the discrete logarithm problem.

Condition 2 comes from the existential forgery attack which enables forging (h, r, s) triplets where h plays the role of SHA-1(M): for DSA, one can just pick $r = (g^\alpha y^\beta \bmod p) \bmod q$ for random α and β, take $s = \frac{r}{\beta} \bmod q$, then $h = s\alpha \bmod q$. (A similar existential forgery attack holds for ECDSA.) With the triplet one can try inverting SHA-1 on h and get a valid signed message.

Condition 3 is trivial: if one can get a collision SHA-1(M_1) = SHA-1(M_2) then one can ask for the signature of M_1 (in a chosen message attack) then forge the signature for M_2.

If Condition 4 does not hold one can predict k then extract the secret key from s and r. □

We can quantify the workload of brute force attacks: Shanks algorithm can break Condition 1 within a complexity $\Omega(\sqrt{q})$ (resp. $\Omega(\sqrt{n})$). Random search can break Condition 2 within 2^{160} computations. The birthday attack breaks Condition 3 within 2^{80} computations. One can also break Condition 4 within 2^{160} trials.

4.2 Brickell Model

In the Brickell model[3], we assume that both SHA-1 and the mod q (resp. $(x_1, y_1) \mapsto \overline{x_1} \bmod n$) function from the subgroup spanned by g (resp. G) behave like random oracles. (Note that it implies Conditions 2 and 3 of Theorem 1.) Under this assumption, Condition 1 and another assumption which may be stronger and Condition 4, one can formally prove that DSA (resp. ECDSA) is secure. We quote the result from [19]. The proof comes from the Forking Lemma techniques of Pointcheval and Stern [18] and is available in [19,8]. The same result holds for ECDSA.

Theorem 2. *In DSA we assume that SHA-1 is replaced by a uniformly distributed random oracle H_1 and the computation of r is replaced by $H_2(g^k \bmod p)$ where H_2 is another uniformly distributed random oracle. We further assume that the pseudorandom generator for k is indistinguishable from a uniformly distributed random generator. Given an algorithm which given the public key y forges a valid (M, h, r, s) quadruplet in time $O(T)$, probability greater than ε, and $O(N)$ oracle accesses, we can make an algorithm which given y computes x within $O(N/\varepsilon)$ replays of the given algorithm.*

We notice that all conditions are necessary but for

– the one on the mod q (resp. $(x_1, y_1) \mapsto \overline{x_1} \bmod n$) function,

[3] This was presented as an invited talk at CRYPTO' 96 but unpublished. See [19,8].

– the one on the pseudorandom generator (which is stronger than Condition 4),
– the random oracle model which is known to be controversial.

The condition on the mod q function is not very satisfactory for DSA. It may look better for ECDSA since the $x_1 \mapsto \overline{x_1}$ is arbitrary.

This model was later generalized by Yung et al. in [8] as the TEGTSS-I scheme.

4.3 Pointcheval-Vaudenay Model

The Pointcheval-Vaudenay result (see [19]) holds with more realistic assumptions for a variant of DSA which is included in the ISO/IEC 14888 [3] standard. In this variant SHA-1(M) is replaced by SHA-1$(r|M)$ as in the Schnorr signature [20,21].

In the Pointcheval-Vaudenay result, SHA-1 still needs to behave like a random oracle. The hypothesis on the mod q (resp. $(x_1, y_1) \mapsto \overline{x_1} \bmod n$) function is replaced by a $O(\log q)$-collision freedom (resp. $O(\log n)$-collision freedom).[4] This means that all preimage sizes are bounded by $O(\log q)$ (resp. $O(\log n)$). This hypothesis is quite realistic (it actually holds for random functions with same range). Therefore all assumptions are realistic, but for the controversy on the random oracle model.

Theorem 3 (Pointcheval-Vaudenay [19,8]). *We consider the Pointcheval-Vaudenay variant of DSA in which SHA-1(M) is replaced by SHA-1$(r|M)$. In this scheme we assume that SHA-1 is replaced by a uniformly distributed random oracle H. The function f over $\mathbf{Z}_q$ which maps k to $f(k) = (g^k \bmod p) \bmod q$ is assumed to be $O(\log q)$-collision free. We further assume that the pseudorandom generator for k is indistinguishable from a uniformly distributed random generator. Given an algorithm which given the public key y forges a valid (M, h, r, s) quadruplet in time $O(T)$, probability greater than ε, and $O(N)$ oracle accesses, we can make an algorithm which given y computes x within $O(N \log q \log\log q/\varepsilon)$ replays of the given algorithm.*

The key idea in the proof is in an improvement of the Forking Lemma which makes forks with $\Omega(\log q)$ branches instead of two.

This model was later generalized by Yung et al. in [8] as the TEGTSS-II scheme. As shown by Lee and Smart [15], the same variant and result can be made for ECDSA.

4.4 Brown Model

Brown [9] presented another proof model in which functions are no longer random oracles, but the underlying group operations are performed in a generic group. One property of generic group is that discrete logarithm is provably hard which implies Condition 1 of Theorem 1. The best attack requires $\Omega(\sqrt{q})$ (resp.

[4] In this paper, t-collision free means that no t inputs collide on their outputs. It is stronger than collision resistance in the sense that collisions really do not exists.

$\Omega(\sqrt{n})$) as proven by Shoup [22]. For DSA, it means that multiplications modulo p are assumed to be represented by a generic group, which is a wrong assumption since sieving algorithms can compute discrete logarithm in sub-exponential time. For ECDSA, it means that elliptic curve points addition is assumed to be represented by a generic group. It is quite natural for random elliptic curves as long as our understanding on the group structure is currently limited. For some special curves like Koblitz curves this hypothesis is not valid at all since some exponential can be computed faster than the regular square-and-multiply algorithm.

Brown also requires the assumption that the $(x_1, y_1) \mapsto \overline{x_1} \bmod n$ function is invertible, which is correct for ECDSA, but we have no similar property for DSA. (One would need that the mod q function of the subgroup spanned by g is invertible which is an open problem so far.) Hence the Brown model is meaningful for ECDSA with random elliptic curves only.

In this model Brown has shown that ECDSA is secure under Conditions 2 and 3 and undistinguishability of the generator for k. His proof was a little flawed, but fixable as pointed out by Stern et al. [24].

Theorem 4. *In ECDSA we assume that elliptic curve is replaced by a generic group of order n greater than 2^{160}. We assume that the $(x_1, y_1) \mapsto \overline{x_1} \bmod n$ function from the generic group to $\mathbf{Z}_n$ is invertible. We further assume that the pseudorandom generator for k is indistinguishable from a uniformly distributed random generator. Given an algorithm which given the public key y forges a valid (M, r, s) triplet in time $O(T)$, probability greater than ε, and $N = O(\sqrt{n\varepsilon})$ oracle accesses, we can make an algorithm which inverts SHA-1 with a complexity of $O((T + N \log N)N/\varepsilon)$ or which finds a collision with complexity $O((T + N \log N)/\varepsilon)$.*

5 Elliptic Curve Validation

The proposed elliptic curve generator of ECDSA works as follows. It consists of selecting first a finite field, second a seed of a pseudorandom generator (denoted seed) which generates a j-invariant, and third the elliptic curve of required j-invariant over the field.

Curves over $\mathbf{F}_p$ with $p > 3$ prime:

1. We choose a prime $p > 3$ and consider $\mathbf{F}_p$.
2. We generate a random bitstring c from seed.
3. We translate c into a field element.
4. We arbitrarily select a and b such that $a^3/b^2 \equiv c \pmod p$ and take the elliptic curve defined by

$$y^2 \equiv x^3 + ax + b \pmod p.$$

We recall that in this case the j-invariant is

$$j = 6912 \frac{a^3/b^2}{4a^3/b^2 + 27} \bmod p = 6912 \frac{c}{4c + 27} \bmod p$$

hence j is fully validated by seed.

Curves over $\mathbf{F}_q$ of characteristic 2:

1. We choose q a power of 2 and consider $\mathbf{F}_q$. We choose a representation of $\mathbf{F}_q$ (i.e. an irreducible polynomial).
2. We generate a random bitstring c from seed.
3. We translate c into a field element and call it b.
4. We arbitrarily select a and take the elliptic curve defined over $\mathbf{F}_q$ by

$$y^2 + xy = x^3 + ax^2 + b.$$

We recall that in this case the j-invariant is $j = \frac{1}{b}$ hence j is fully validated by seed.

Then the seed is kept as a certificate of good forgery. Users can check that seed validates the j-invariant of the curve. This certificate is assumed to convince that one hid no trapdoor in the specific choice of the curve.

5.1 Elliptic Curves with Same j-Invariant

As noticed, only j is validated from seed. The j invariant is however not a complete characteristic for the elliptic curve. One can replace an elliptic curve by its twist which has the same j-invariant but is not isomorphic. In the prime field case, we can replace (a, b) by $(u^2 a \bmod p, u^3 b \bmod p)$ where u is a non-quadratic residue in $\mathbf{F}_p$. In characteristic two fields, we can replace (a, b) by $(a + \theta, b)$ where θ is an element of trace 1 over $\mathbf{F}_2$, i.e. $\mathrm{Tr}(\theta) = 1$ where Tr is defined by

$$\mathrm{Tr}(\theta) = \theta + \theta^2 + \theta^{2^2} + \theta^{2^3} + \ldots + \theta^{\frac{q}{2}}.$$

Both curves have the same j-invariant. Although they are isomorphic in some extension field, they are not isomorphic in general over the chosen field. Therefore the exact choice of the curve is not certified by this scheme. This means that if users accept an elliptic curve with the field/seed certificate, they also implicitly accept its twist.

This property can easily be avoided by requiring that seed generates c together with an extra bit which is such that

- $\left(\frac{b}{p}\right) = (-1)^{\mathrm{bit}}$ for prime fields,
- $\mathrm{Tr}(a) = \mathrm{bit}$ for characteristic two fields.

In this case seed fully validates the elliptic curve up to an isomorphism.

5.2 Bypassing the Scheme in the Characteristic Two Case

In the standard validation scheme, the pseudorandom generator for c is used *after* the finite field choice, though it does not really use it. One can wonder what happens if we first generate c then choose the finite field. The characteristic two case is easy since the field representation choice is open, and basically free.

Hence one can easily bypass the validation scheme as follows. We assume that we have an elliptic curve defined by

$$y^2 + xy = x^3 + ax^2 + b$$

over a given field $\mathbf{F}_q$ of characteristic two. We want to validate it by looking for a field representation and seed.

1. Pick seed at random.
2. Generate a bitstring c from seed.
3. Look for the field representation of $\mathbf{F}_q$ such that the string c represents the field element b. If not possible go back to step 1.

As shown in Appendix A, this works within less than $\log_2 q$ iterations on average for any b but $b = 0$ (for which the curve is singular) or $b = 1$ (which is quite relevant for Koblitz curves). Note that Step 3 simply consists of looking for roots of the polynomial equation $C(X) = b$ where $C(X)$ is defined by c. (See Appendix A.)

Therefore the pseudorandom generation of the elliptic curve provided in ANSI X9.62 [2] brings a very weak guaranty of honest elliptic curve generation.

5.3 Prime Fields Case

The prime field case do not offer any choice for the field representation. However we can still try to choose p after having generated c. At the time this paper is written we have no clue how to maliciously pick p so that the elliptic curve is weak. We let this as an open problem.

> *Given a random integer c, can we choose a prime p such that an elliptic curve over $\mathbf{F}_p$ whose j-invariant is $j = 6912\frac{c}{4c+27} \bmod p$ is flawed?*

As a challenge we propose

> *Given a random integer c, can we choose a prime p such that $j = 6912\frac{c}{4c+27} \bmod p$ is the j-invariant of an anomalous elliptic curve over $\mathbf{F}_p$?*

Anomalous elliptic curves over $\mathbf{F}_p$ are known to be weak.[5]

5.4 A Possible Tweak for ECDSA

We recommend to update the validation scheme by concatenating the seed with p (resp. q and the field representation) in the generator and by generating an extra bit. Here is the tweaked scheme.

Curves over $\mathbf{F}_p$ with $p > 3$ prime:

1. We choose a prime $p > 3$ and consider $\mathbf{F}_p$.

[5] See [23,17].

2. We generate a random bitstring c and bit from seed and p.
3. We translate c into a field element.
4. We arbitrarily select a and b such that $a^3/b^2 \equiv c \pmod{p}$ and $\left(\frac{b}{p}\right) = (-1)^{\mathrm{bit}}$, and take the elliptic curve defined by a, b, p.

Curves over $\mathbf{F}_q$ of characteristic 2:

1. We choose q a power of 2 and consider $\mathbf{F}_q$. We choose a representation of $\mathbf{F}_q$ (i.e. an irreducible polynomial).
2. We generate a random bitstring c and bit from seed, q, and the field representation choice.
3. We translate c into a field element and call it b.
4. We arbitrarily select a such that $\mathrm{Tr}(a) = \mathrm{bit}$ and take the elliptic curve defined over $\mathbf{F}_q$ by a, b, q.

This way we are ensured that the elliptic curve was randomly selected up to an isomorphism.

6 Conclusion

We surveyed security properties of DSA and ECDSA.

- We have seen that, like for many cryptographic schemes, DSA and ECDSA are highly vulnerable when used in a poor way. For instance the pseudorandom generator must be cryptographically strong. We also need to care about cloning issues.
- We also investigated the parameter validation issue. Like the public key validation problem, parameter must be validated and securely stored. Otherwise one can hide trapdoors in p, q, or g (resp. G).
- DSA and ECDSA are provably secure in the random oracle model by assuming that the $k \mapsto r$ has nice properties. Slight variants of DSA and ECDSA benefit from stronger security proofs.

Finally, the standard elliptic curve validation scheme happens to provide weak guaranty for honest generation. While the prime field case is proposed as an open problem, the validation process is easy to bypass in the characteristic two case. Therefore we recommend to update the elliptic curve validation scheme as proposed in Section 5.4.

7 Acknowledgments

I would like to thank Japanese Government (IPA) for having proposed this work under the CRYPTREC project.[6] I would also like to thank Daniel Bleichenbacher for providing information on his attack and Franck Leprevost for helpful discussions.

[6] `http://www.ipa.go.jp/security/index-e.html`

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. ANSI X9.62. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). American National Standard Institute. American Bankers Association. 1998.
3. ISO/IEC 14888. Information Technology — Security Techniques — Digital Signatures with Appendix. ISO/IEC, Geneva, Switzerland, 1998.
4. Secure Hash Standard. *Federal Information Processing Standard* publication #180-1. U.S. Department of Commerce, National Institute of Standards and Technology, 1995.
5. Digital Signature Standard (DSS). *Federal Information Processing Standards* publication #186-2. U.S. Department of Commerce, National Institute of Standards and Technology, 2000.
6. 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.
7. 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.
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. T. ElGamal. Cryptography and Logarithms over Finite Fields. PhD Thesis, Stanford University, 1984.
11. 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.
12. 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.
13. N. Koblitz. CM-Curves with good Cryptographic Properties. In *Advances in Cryptology CRYPTO'91*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 576, pp. 279–287, Springer-Verlag, 1992.
14. R. Lidl, H. Niederreiter. *Introduction to Finite Fields and their Applications*, Revised Edition, Cambridge University Press, 1994.
15. J. Malone-Lee, N. P. Smart. Modifications of ECDSA. To appear in the proceedings of SAC'02.
16. U. Maurer, S. Wolf. Lower Bounds on Generic Algorithms in Groups. In *Advances in Cryptology EUROCRYPT'98*, Espoo, Finland, Lectures Notes in Computer Science 1403, pp. 72–84, Springer-Verlag, 1998.
17. J. Monnerat. Computation of the Discrete Logarithm on Elliptic Curves of Trace One — Tutorial. Technical report IC 200249, EPFL, 2002.
`http://lasecwww.epfl.ch`

18. D. Pointcheval, J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, vol. 13, pp. 361–396, 2000.
19. D. Pointcheval, S. Vaudenay. On Provable Security for Digital Signature Algorithms. Technical report LIENS 96-17, Ecole Normale Supérieure, 1996.
20. 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.
21. C. P. Schnorr. Efficient Identification and Signature for Smart Cards. *Journal of Cryptology*, vol. 4, pp. 161–174, 1991.
22. V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In *Advances in Cryptology EUROCRYPT'97*, Konstanz, Germany, Lectures Notes in Computer Science 1233, pp. 256–266, Springer-Verlag, 1997.
23. N. P. Smart. The Discrete Logarithm Problem on Elliptic Curves of Trace One. *Journal of Cryptology*, vol. 12, pp. 193–196, 1999.
24. J. Stern, D. Pointcheval, J. Malone-Lee, N. P. Smart. Flaws in Applying Proof Methodologies to Signature Schemes. In *Advances in Cryptology CRYPTO'02*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 2442, pp. 93–110, Springer-Verlag, 2002.
25. 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.

A Number of Field Representations

Theorem 5. *Let ℓ be an integer. Let $q = 2^\ell$ and consider $\mathbf{F}_q$. Let $b \in \mathbf{F}_q$. Let $C(X) = c_0 + c_1X + \ldots + c_{\ell-1}X^{\ell-1} \in_U \mathbf{F}_2[X]$ be a uniformly distributed random binary polynomial of degree at most $\ell - 1$. We consider the probability* Pr *that there exists an element $x \in \mathbf{F}_q$ of degree ℓ such that $C(x) = b$.*

- *For $b = 0$ or $b = 1$, we have $\Pr = 2^{-\ell}$.*
- *In other cases, we have $\Pr \geq \frac{1}{\ell}$ for $\ell \neq 6$, and $\Pr \geq \frac{1}{7}$ for $\ell = 6$.*

Given an element $x \in \mathbf{F}_q$ of degree ℓ, we know that $1, x, x^2, \ldots, x^{\ell-1}$ is a basis of $\mathbf{F}_q$ over $\mathbf{F}_2$ so we can choose the minimal polynomial $\mu_x(X)$ of x in order to represent $\mathbf{F}_q$ as $\mathbf{F}_2[X]/\mu_x(X)$. This way C represents b.

This theorem thus means that for any b which is neither 0 nor 1 and any bitstring C there is a probability greater than $\frac{1}{\ell}$ that there exists a representation of $\mathbf{F}_q$ in which C represents b.

Proof. When $b = 0$ or $b = 1$, we notice that $C(x) = b$ implies that x is a root of $C(X) - b$ which is a binary polynomial. When this polynomial is nonzero, this means that x is a root of a polynomial of degree less than ℓ, so x cannot have a degree of ℓ. Hence x exists only when $C(X)$ is identically equal to b which holds with probability $2^{-\ell}$. Let us now consider other cases.

Let A be the set of all $x \in \mathbf{F}_q$ of degree ℓ. For any $x \in A$, we know that $1, x, x^2, \ldots, x^{\ell-1}$ is a basis of $\mathbf{F}_q$ over $\mathbf{F}_2$. Thus we can represent b as a linear combination and obtain $b = C(x)$ for some polynomial $C(X)$ of degree less than ℓ. Hence to any $x \in A$ corresponds a unique polynomial which we denote $C_x(X)$.

For any polynomial $C(X) = c_0 + c_1X + \ldots + c_{\ell-1}X^{\ell-1}$ in $\mathbf{F}_2[X]$ the values $x \in A$ such that $C_x(X) = C(X)$ are all roots of $C(X) - b$ which is a *nonzero* polynomial in $\mathbf{F}_q[X]$ of degree at most $\ell - 1$. Hence we have at most $\ell - 1$ values $x \in A$ which are mapped onto the same $C_x(X)$. Hence the number of $C(X)$ for which we have a solution is at least $\frac{|A|}{\ell-1}$ where $|A|$ is the cardinality of A. Thanks to the following lemma we obtain that $\Pr \geq \frac{1}{\ell-1}\left(1 - \frac{1}{\ell}\right) = \frac{1}{\ell}$ for $\ell \neq 6$. For $\ell = 6$ we have $\Pr \geq \frac{1}{5}\left(1 - \frac{11}{64}\right) \geq \frac{1}{7}$. □

Lemma 6. *Let ℓ be an integer. Let $q = 2^\ell$ and consider $\mathbf{F}_q$. The number N of $x \in \mathbf{F}_q$ of degree ℓ is such that $N \geq q - \sqrt{q}\log_2 \ell$. For $\ell \neq 6$ we also have $N \geq q - \frac{q}{\ell}$ which is tighter for $\ell \leq 10$. For $\ell = 6$ we have $N = q - 11$.*

Proof. We proceed by upper bounding the number of x whose degree is less than ℓ. Let $\ell = p_1^{\alpha_1} \ldots p_r^{\alpha_r}$ be the factorization of ℓ. We assume that the p_is are pairwise different prime integers and that the α_is are non negative integers.

Let $x \in \mathbf{F}_q$ of degree $d < \ell$. The minimal polynomial $\mu_x(X)$ of x is an irreducible polynomial of $\mathbf{F}_2[X]$ of degree d whose roots are the conjugates of x. The $\mathbf{F}_2[X]/\mu_x(X)$ finite field contains the roots of $\mu_x(X)$ and is thus a subfield of $\mathbf{F}_q$ and d divides ℓ. We deduce that $x^{2^d} = x$. Since $d < \ell$ and d divides ℓ there must exists i such that d divides $\frac{\ell}{p_i}$. Raising to the power $2^{\frac{\ell}{p_i}}$ is equivalent to raising $\frac{\ell}{dp_i}$ times to the power 2^d. Hence x is a root of $X^{2^{\frac{\ell}{p_i}}} - X$. We deduce that the number of $x \in \mathbf{F}_q$ of degree $d < \ell$ is at most $\sum_{i=1}^{r} 2^{\frac{\ell}{p_i}}$. Since $p_i \geq 2$ we have $2^{\frac{\ell}{p_i}} \leq \sqrt{q}$. Similarly we have $r \leq \log_2 \ell$ so the number of elements of degree less than ℓ is at most $\sqrt{q}\log_2 \ell$.

In order to deduce a tighter bound we use the explicit number of irreducible polynomial of degree ℓ over $\mathbf{F}_2[X]$ which is

$$\frac{1}{\ell} \sum_{d \text{ divides } \ell} \mu(d) 2^{\frac{\ell}{d}}$$

where μ is the Mœbius function (see [14, pp. 84–86]). To each irreducible polynomial corresponds ℓ roots so we have

$$N = \sum_{d \text{ divides } \ell} \mu(d) 2^{\frac{\ell}{d}}$$

which leads us to

$$N = \sum_{\beta_1=0}^{1} \ldots \sum_{\beta_r=0}^{1} (-1)^{\beta_1+\ldots+\beta_r} \times 2^{\ell/(p_1^{\beta_1} \ldots p_r^{\beta_r})}.$$

We can check that $N \geq q - \frac{q}{\ell}$ for all $\ell \leq 10$ but $\ell = 6$ for which $N = 2^6 - 2^3 - 2^2 + 2^1 = 2^6 - 11$.

We easily demonstrate that for $\ell \geq 11$ we have $\sqrt{q}\log_2 \ell \leq \frac{q}{\ell}$ so $N \geq q - \frac{q}{\ell}$ holds for all ℓ but $\ell = 6$. □