Lecture 24

A Quantum Algorithm for Breaking Digital Signatures

of the course "Fundamentals of Quantum Computing"
(by IAPS and QUANTERALL)

Stoyan Mishev, Vesselin Gueorguiev and Vladimir Gerdjikov

INSTITUTE *for* ADVANCED PHYSICAL STUDIES

NEW BULGARIAN UNIVERSITY

January 13, 2022

Plan

DSA protocol

Hashing

Elliptic functions

Public Key and Signature Generation, Verification

DSA protocol

Digital Signature Algorithm (DSA) Protocol	
Signer	**Public knows**
	algorithm parameters $\mathcal{A}$
	verification statement v
chooses a private key k	
creates a public *verification key* by	
computing a $V = V(k, \mathcal{A})$	
and publishing it	verification key V
signs document by	
taking document d,	document d
computing a *signature* $s(d, \mathcal{A})$	
and publishing it	signature s
	and can verify by
	checking the verification statement $v(s, d, V, \mathcal{A}) = \text{TRUE}$?

Example 6.19 An example of a hash function provided by the NSA is the Secure Hashing Algorithm SHA256 which converts any ASCII into a 64 digit hexadecimal string. As an example consider the following text.
The SHA256 hash output of the text in this line in hexadecimal form displayed across two lines is:

$$A3C431026DDD514C6D0C7E5EB253D424$$
$$B6A4AF20EC00A8C4CBE8E57239BBB848$$

Such a 64 digit hexadecimal string can be interpreted as a 256-bit natural number d, which in our example would be (given in binary format first)

$$
\begin{aligned}
d \\
=&(10100011110001000011000100000010011011011101110101010000000\\
&00000000000000000000000000000000000000000000000000000000000\\
&00000000000000000000000000000000000000000000000000000000000\\
&00000000000000000000000000000000000000000000000000000000000\\
&00000000000000000000)_2\\
=&7.407363459482995\cdots \times 10^{76} < 2^{256}.
\end{aligned}
$$

A widely used version of such a DSA is based on the difficulty to find discrete logarithms for elements of elliptic curves (ECDSA). ECDSAs are usually based on elliptic curves $E(F_p)$ for which p is a large prime. For a prime p the elliptic curve $E(F_p)$ over the finite field $F_p = Z/pZ$ together with the addition $+_E$ given in Theorem F.58 forms a finite abelian group

A widely used version of such a DSA is based on the difficulty to find discrete logarithms for elements of elliptic curves (ECDSA). ECDSAs are usually based on elliptic curves $E(F_p)$ for which p is a large prime. For a prime p the elliptic curve $E(F_p)$ over the finite field $F_p = Z/pZ$ together with the addition $+_E$ given in Theorem F.58 forms a finite abelian group

ECDSA Parameters (p, A, B, P, q) **in the Public Domain**

1. A prime p specifying the finite field $\mathbb{F}_p$
2. Two elements $A, B \in \mathbb{F}_p$ specifying the WEIERSTRASS equation

$$y^2 = x^3 + Ax + B$$

   of the elliptic curve $E(\mathbb{F}_p)$. This is an equation in the finite field $\mathbb{F}_p$. The underlying set of $\mathbb{F}_p$ consists of cosets in $\mathbb{Z}/p\mathbb{Z} \cong \mathbb{Z}_p$. From Lemma F.5 and Example F.19 we know that any such coset (or equivalently element in $\mathbb{Z}_p$) can be uniquely identified with a number in $\{0, \ldots, p-1\}$. Hence, we consider A and B and the components x and y of elements $P = (x, y) \in E(\mathbb{F}_p) \setminus \{0_E\}$ as elements of the set $\{0, \ldots, p-1\}$
3. An element

$$P = (x_P, y_P) \in E(\mathbb{F}_p) \setminus \{0_E\} \subset \mathbb{F}_p \times \mathbb{F}_p,$$

   which is often called the base point of the ECDSA

4. The element P is chosen such that it has prime order, that is,

$$q = \operatorname{ord}(P) := \min\left\{n \in \mathbb{N} \,\middle|\, nP = 0_E \in E(\mathbb{F}_p)\right\}$$

is a publicly known prime

ECDSA Public Key Generation

1. Select a private key

$$k \in \{1, \ldots, q-1\} \subset \mathbb{N}$$

2. Compute the *verification key*

$$V = kP \in E(\mathbb{F}_p) \setminus \{0_E\}.$$

   Note that $V \neq 0_E$ since $k < q$, and q is the smallest number such that $qP = 0_E$
3. Publish the verification key $V \in E(\mathbb{F}_p) \setminus \{0_E\}$

ECDSA Signature Generation

1. Select a natural number

$$a \in \{1, \ldots, q-1\}$$

2. Compute

$$aP = (x_{aP}, y_{aP}) \in E(\mathbb{F}_p) \setminus \{0_E\},$$

where, as above, we are guaranteed $aP \neq 0_E$ since $a < q$, and we consider $x_{aP} \in \mathbb{F}_p$ to be represented by a number in $\{0, \ldots, p-1\}$

3. Compute

$$s_1 = x_{aP} \bmod q \in \{0, \ldots, q-1\}$$

4. If $s_1 = 0$, go back to Step 1 of the signature generation and select a new $a \in \{1, \ldots, q-1\}$.
   If $s_1 \neq 0$, calculate the multiplicative inverse of a modulo q

$$\widehat{a} = a^{-1} \bmod q \in \{1, \ldots, q-1\}$$

   defined in Definition D.8, that is, the number $\widehat{a}$ such that $a\widehat{a} \bmod q = 1$. Note that since $a \in \{0, \ldots, q-1\}$ and q is a prime, we always have $\gcd(a, q) = 1$ and the multiplicative inverse exists.
   With $\widehat{a}$ compute

$$s_2 = \big((d + ks_1)\widehat{a}\big) \bmod q \in \{0, \ldots, q-1\}$$

5. If $s_2 = 0$, go back to Step 1 of the signature generation and select a new $a \in \{1, \ldots, q-1\}$.
   Else, set the *signature* as

$$(s_1, s_2) \in \{1, \ldots, q-1\} \times \{1, \ldots, q-1\}$$

6. Publish the signature (s_1, s_2)

ECDSA Verification

1. Compute

$$\widehat{s_2} = s_2^{-1} \bmod q$$
$$u_1 = d\widehat{s_2} \bmod q$$
$$u_2 = s_1\widehat{s_2} \bmod q$$

   and with these calculate

$$(x, y) = u_1 P + u_2 V$$

2. Check if

$$x \bmod q \stackrel{?}{=} s_1$$

   is true. If it is, then (s_1, s_2) constitutes a valid signature of the document d. Otherwise, it does not

Elliptic Curve Digital Signature (ECDSA) Protocol	
Signer	**Public knows**
	algorithm parameters $\mathcal{A}$: large prime p elliptic curve $E(\mathbb{F}_p)$ public point $P \in E(\mathbb{F}_p) \setminus \{0_E\}$ with a large prime order q
creates key by choosing a *secret signing key* $k \in \mathbb{N}$ with $1 < k < q$, computing the *verification key* $V = kP$	
and publishing it	verification key V
signs document by	
taking document d and a random $a \in \mathbb{N}$ with $a < q$,	document d
computing $aP \in E(\mathbb{F}_p) \setminus \{0_E\}$ $s_1 = x_{aP} \bmod q$ $s_2 = \big((d + ss_1)(a^{-1} \bmod q)\big) \bmod q$	
and publishing the *signature* (s_1, s_2)	signature (s_1, s_2)
	and verifies by computing $u_1 = \big(d(s_2^{-1} \bmod q)\big) \bmod q$ $u_2 = \big(s_1(s_2^{-1} \bmod q)\big) \bmod q$ $(x, y) = u_1 P +_E u_2 V \in E(\mathbb{F}_p) \setminus \{0_E\}$

In Definition 6.17 we defined for any $V, P \in E(\mathbb{F}_p)$ such that $V = kP$

$$k = \mathrm{dlog}_P(V)$$

as the **discrete logarithm** in $E(\mathbb{F}_p)$ of V to base P. The security of ECDSA depends on the fact that it is very hard to calculate the discrete logarithm for this group.

Example 6.21 Bitcoins use the `secp256k1` ECDSA [93] protocol with the WEIERSTRASS equation defined by $A = 0$ and $B = 7$, that is,

$$y^2 = x^3 + 7\,,$$

the prime

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 \tag{6.136}$$

and the public point $P = (x_P, y_P)$ given by

$$\begin{aligned} x_P &= 55066263022277343669578718895168534326250603453777594175500187360389116729240 \\ y_P &= 32670510020758816978083085130507043184471273380659243275938904335757337482424\,. \end{aligned}$$

The best known classical method to calculate $k = \mathrm{dlog}_P(V)$ for $E(\mathbb{F}_p)$ requires $O(\sqrt{p})$ computational steps and thus for the bitcoin ECDSA of the order of $O(10^{77})$ computational steps. In contrast, a quantum computer could potentially calculate $k = \mathrm{dlog}_P(V)$ for $E(\mathbb{F}_p)$ requiring only

$$O(\text{polynomial in } \log(p)) = O(\text{polynomial in } 256)$$

THANK YOU FOR YOUR ATTENTION!

БЛАГОДАРЯ ЗА ВНИМАНИЕТО!