Cryptology Fall 2017

Chloe Martindale
TU/e

September 28, 2017

These notes are based on notes by Tanja Lange and Ruben Niederhagen. Following on from last weeks lecture on finite fields, we now see how to use finite fields in cryptography.

1 Diffie-Hellman key exchange

Suppose that Alice and Bob want to compute a shared secret via the Diffie-Hellman key exchange. They first agree (in public) on a finite field $\mathbb{F}_q = \mathbb{F}_{p^r}$ (so $\mathbb{F}_q = \mathbb{F}_p[x]/f(x)\mathbb{F}_p[x]$ where $f(x) \in \mathbb{F}_p[x]$ is a monic irreducible polynomial of degree r) and a primitive element g of $\mathbb{F}_q^*$ (so $\mathbb{F}_q^* = \langle g \rangle = \{g, \ldots, g^{q-1}\}$). Then

1. Alice chooses a random private key $a \in \{1, \ldots, q-1\}$ and Bob chooses a random private key $b \in \{1, \ldots, q-1\}$.
2. Alice computes her public key $a' = g^a \in \mathbb{F}_q$ and Bob computes his public key $b' = g^b \in \mathbb{F}_q$.
3. Alice and Bob do a *key exchange*, that is, Alice sends a' to Bob and Bob sends b' to Alice.
4. Alice computes $(b')^a = (g^b)^a$ and Bob computes $(a')^b = (g^a)^b$.

Alice and Bob now have a *shared secret key* g^{ab} with which they can encrypt their messages. The safety of this protocol relies of the hardness of the *discrete logarithm problem* in $\mathbb{F}_q$, that is, on the hardness of computing $a \in \{1, \ldots, q-1\}$ given $g^a \in \mathbb{F}_q$. This problem is believed to be hard for very large q, a, and b. Even given the hardness of the discrete logarithm problem, the Diffie-Hellman key exchange is still vulnerable to man-in-the-middle attacks, has no authentication checks, ...

2 ElGamal encryption

Suppose that Bob wants to encrypt a message m and send it to Alice. They can use *ElGamal encryption*, which has 3 parts: setup, encryption, and decryption.
Setup:

1. Alice chooses a finite field $\mathbb{F}_q$, a primitive element g of $\mathbb{F}_q^*$.
2. Alice chooses a random private $a \in \{1, \dots, q-1\}$ and computes $h = g^a$.
3. Alice sends her public key $(\mathbb{F}_q, g, h)$ to Bob.

Encryption:

1. Bob choose a random private $b \in \{1, \dots, q-1\}$ and computes $c_1 = g^b$.
2. Bob computes the shared secret $s = h^b = (g^a)^b = g^{ab}$.
3. Bob computes $c_2 = ms$.
4. Bob sends the ciphertext (c_1, c_2) to Alice.

Decryption:

1. Alice computes the shared secret $s = c_1^a = (g^b)^a = g^{ab}$.
2. Alice decrypts the ciphertext $m = c_2 s^{-1} = c_2 c_1^{q-1-a}$.

Observations.

- Note that $s^{-1} = c_1^{q-1-a}$ as $sc_1^{q-1-a} = g^{ab} g^{b(q-1-a)} = (g^b)^{q-1} = 1$.
- If m is known, the shared secret g^{ab} can be recovered from the ciphertext: use a new b for each message!

3 ElGamal signatures

Suppose that we are given a database with a hash function $H : \{0,1\}^* \to \mathbb{Z}$, a finite field $\mathbb{F}_q$ and a primitive element g of $\mathbb{F}_q^*$. We can then choose a random private key $a \in \{1, \dots, q-1\}$ and create a public key $g^a \in \mathbb{F}_q$ for a digital signature. A digital signature algorithm should consist of a sign step and a verify step:
Sign a message m:

1. Pick a random nonce ('number that you use once') $k \in \{1, \dots, q-1\}$.
2. Compute $r = g^k \in \mathbb{F}_q$.
3. Compute $s \equiv k^{-1}(H(m) - ar) \bmod q-1$.
4. Publish signature (r, s).

Verify signature (r, s):

1. Check whether $g^{H(m)} = h^r r^s$.

Note that the verification step works as

$$h^r r^s = g^{ar}(g^k)^s = g^{ar} g^{H(m)-ar} = g^{H(m)}.$$

Observations. 1. Suppose that for some message m, signed by (r, s), the nonce k is discovered. Then

$$a \equiv \frac{H(m) - ks}{r} \bmod q - 1,$$

so the secret key can be recovered!

2. Suppose that the same k is used for two separate messages m_1 and m_2, giving signatures (r, s_1) and (r, s_2) where

$$s_i \equiv k^{-1}(H(m_i) - ar) \bmod q - 1.$$

Then

$$s_1 - s_2 \equiv k^{-1}(H(m_1 - ar) - k^{-1}(H(m_2) - ar) \quad \equiv k^{-1}(H(m_1) - H(m_2)),$$

hence

$$k \equiv \frac{H(m_1) - H(m_2)}{s_1 - s_2} \bmod q - 1.$$

So k can be recovered, and the secret key is then recoverable by (1). So never reuse your nonce!

3. The signature depends only on $H(m)$, not on m, so in theory an attacker could find another message m' such that $H(m) = H(m')$, as the signature for m and for m' would be the same. In practise however this doesn't happen since cryptographic hash functions are designed to be 'collision resistant', that is, such an m' is unlikely to exist.

4 The Discrete Logarithm Problem

All the protocols above rely on the discrete logarithm problem being hard enough. But how hard is the discrete logarithm problem? Let's do a small example.

Example. Suppose that we want to solve the discrete logarithm problem for $q = 7$, $g = 3$, and $h = 5$. That is, we want to find an integer a with $1 \leq a \leq 6$ such that $3^a = 5 \bmod 7$. (Note that 3 is a primitive element for $\mathbb{F}_7^*$.) To do this, we can just try different a until we succeed:

$$3^1 \equiv 3, \quad 3^2 \equiv 2, \quad 3^3 \equiv 6, \quad 3^4 \equiv 4, \quad 3^5 \equiv 5,$$

so $a = 5$.

We see from the example that the discrete logarithm is trivial for small numbers, but if we try to solve it in this way for q of cryptographic size then it is not feasible. Let's now consider a slightly bigger example in which we can do something slightly more clever:

Example. Suppose that we want to solve the discrete logarithm problem for $q = 37$, $g = 2$, and $h = 17$. That is, we want to find an integer a with $1 \leq a \leq 36$ such that $2^a \equiv 17 \bmod 37$. (Here 2 is a primitive element of $\mathbb{F}_{37}^*$.) Remember from last week that $\mathbb{F}_{37}^*$ is a cyclic group with 36 elements, so in particular it has cyclic subgroups of size 2, 3, 4, 6, 9, 12, and 18. We use this structure to compute a mod 2, 3, etc. until we have information about a to deduce it via the Chinese Remainder Theorem.

- We start by computing a mod 2. The subgroup of $\langle 2 \rangle = \mathbb{F}_{37}^*$ of order 2 is given by $\langle 2^{18} \rangle$. Write $a = a_0 + 2a_1$ (so that $a \equiv a_0 \bmod 2$). Then $2^a \equiv 17 \bmod 37$, so

  $$-1 \equiv 17^1 8 \equiv (2^a)^{18} \equiv 2^{18a_0+36a_1} \equiv 2^{18a_0} 2^{36a_1} \equiv (2^{18})^{a_0} = (-1)^{a_0}.$$

  Hence $a_0 = 1$ and therefore

  $$a \equiv 1 \bmod 2.$$

- We now compute a mod 3. The subgroup of $\langle 2 \rangle = \mathbb{F}_{37}^*$ of order 3 is given by $\langle 2^{12} \rangle$. Write $a = b_0 + 3b_1$ (so that $a \equiv b_0 \bmod 3$). Then $2^a \equiv 17 \bmod 37$, so

  $$26 \equiv 17^{12} \equiv (2^a)^{12} \equiv 2^{12b_0+36b_1} \equiv 2^{12b_0} 2^{36b_1} \equiv (2^{12})^{b_0} \equiv 26^{b_0}.$$

  Hence $b_0 = 1$ and therefore

  $$a \equiv 1 \bmod 3.$$

- We now compute a mod 4. The subgroup of $\mathbb{F}_{37}^*$ of order 4 is given by $\langle 2^9 \rangle$. We know from (1) that $a = 1 + 2a_1$, so it suffices to know a_1 mod 2. Write $a_1 = c_0 + 2c_1$ (so that $a_1 \equiv c_0 \bmod 2$). Note that we know the value of 2^{2a_1} as

  $$17 \equiv 2^a \equiv 2^{1+2a_1} \equiv 2 \cdot 2^{2a_1},$$

  so $2^{2a_1} \equiv 2^{-1} \cdot 17 \equiv 27$. This then gives us that

  $$-1 \equiv 27^9 \equiv (2^{2a_1})^9 \equiv (2^{2c_0+4c_1})^9 \equiv 2^{18c_0} 2^{36c_1} \equiv (-1)^{c_0}.$$

  Hence $c_0 = 1$ and therefore

  $$a \equiv 3 \bmod 4.$$

- We now compute a mod 9. The subgroup of $\mathbb{F}_{37}^*$ of order 9 is given by $\langle 2^4 \rangle$.We know from (2) that $a = 1 + 3b_1$, so it suffices to know b_1 mod 3. Write $b_1 = d_0 + 3d_1$ (so that $b_1 \equiv d_0 \bmod 3$). Note that we know the value of 2^{3b_1} as

  $$17 \equiv 2^a \equiv 2^{1+3b_1} \equiv 2 \cdot 2^{3b_1},$$

so $2^{3b_1} \equiv 2^{-1} \cdot 17 \equiv 27$. This then gives us that

$$10 \equiv 27^4 \equiv (2^{3b_1})^4 \equiv 2^{(12d_0+36d_1)} \equiv (2^{12})^{d_0} \equiv 26^{d_0}. \equiv$$

Hence $d_0 = 2$ and therefore

$$a \equiv 7 \bmod 9.$$

Now (3) and (4) together with the Chinese Remainder Theorem tell us that $a \equiv 7 \bmod 36$, hence our discrete logarithm problem is solved.

From the example above we see that it if fact suffices to solve the discrete logarithm problem in every prime power order subgroup of $\mathbb{F}_q^*$ and then use the Chinese remainder theorem, which can be much more efficient in the right situation. The generalisation of the above example is called the *Pohlig-Hellman attack*.

5 Pohlig-Hellman attack

The Pohlig-Hellman attack is an algorithm that outputs $a = \log_g(h)$ given $\mathbb{F}_q$ with primitive element g and some $h \in \mathbb{F}_{q^*}$. It works as follows: factorise $q-1$ as

$$q - 1 = p_1^{\ell_1} \cdots p_r^{\ell_r},$$

where for $1 \leq i < j \leq r$, $p_i \neq p_j$ and primes and for all $i \geq 1$, $\ell_i \geq 1$. Then for each i,

1. Write
$$a = a_{i,0} + a_{i,1}p_i + \cdots a_{i,\ell_i}p_i^{\ell_i},$$
where $0 \leq a_{i,j} < p_i$.
2. Compute $g_i = g^{\frac{q-1}{p_i}}$.
3. Compute $a_{i,0} = \log_{g_i}(h^{\frac{q-1}{p_i}})$ and set $h_0 = h$.
4. For $j = 1, \ldots, \ell_i - 1$,
   (i) compute $h_j = h_{j-1}/g^{a_{i,j-1}p_i^{j-1}}$,
   (ii) compute $a_{i,j} = \log_{g_i} h_j^{\frac{q-1}{p_i^{j+1}}}$.
5. Return $a \bmod p_i^{\ell_i}$.

The Chinese Remainder Theorem then gives $a \bmod q-1$.

To protect against the Pohlig-Hellman attack, we should always choose q so that $\mathbb{F}_q^*$ has a large prime order subgroup (for example, $q = 2^n + 1$ would be very bad)! Some protocols actually work in this prime order subgroup.

Example. DSA, Digital Signature Algorithm, can be implemented in some prime order subgroup $\langle g \rangle \subseteq \mathbb{F}_q^*$ of $\mathbb{F}_q^*$. (NB g is *not* a primitive element of $\mathbb{F}_q^*$ here!) If $\operatorname{ord}(g) = \ell$, the protocol becomes:
Setup:

- Choose a random secret $a \in \{1, \ldots, \ell\}$.
- Compute public key $h = g^a$.

Sign a message m:

- Pick a random nonce k in $\{1, \ldots, q-1\}$ and compute $r' = g^k \in \mathbb{F}_q^*$. Choose $r \in \mathbb{Z}$ such that $r \equiv r' \bmod \ell$.
- Compute $s \equiv k^{-1}(H(m)+ar) \bmod \ell$ (with a pre-determined hash function H).
- The signature is given by (r, s).

Verify:

- Compute:

$$\begin{aligned} w &\equiv s^{-1} \bmod \ell \\ u_1 &\equiv H(m)w \bmod \ell \\ u_2 &\equiv rw \bmod \ell \\ v' &\equiv g^{u_1} h^{u_2} \in \mathbb{F}_q^*. \end{aligned}$$

- Choose $v \in \mathbb{Z}$ such that $v \equiv v' \bmod \ell$.
- Verify that $v \equiv r \bmod \ell$.

Note that this verification does indeed check that (r, s) is a signature for m as

$$v \equiv v' \equiv g^{u_1} h^{u_2} \equiv g^{H(m)w} h^{rw} \equiv g^{w(H(m)+ar)} \equiv g^r \equiv r.$$

So assuming that we choose our prime (power) q such that $\mathbb{F}_q^*$ has a large prime order subgroup, how can we solve the discrete logarithm problem? Pohlig-Hellman solves it in time $O(\ell)$, where ℓ is the size of the largest prime order subgroup, but in fact we can still do better.

6 Baby Step Giant Step

One of the best generic algorithms known for solving the discrete logarithm problem is baby step giant step. Again, we assume that we have a subgroup $\langle g \rangle \subseteq \mathbb{F}_q^*$ of prime order ℓ, and that given $h \in \langle g \rangle$, we want to find $a = \log_g(h)$. The baby step giant step algorithm is as follows:

1. For i from 0 to $\lfloor \sqrt{\ell} \rfloor$, compute $b_i = g^i$.

2. For j from 0 to $\lfloor\sqrt{\ell}\rfloor + 1$, compute $c_j = h \cdot g^{-\lfloor\sqrt{\ell}\rfloor \cdot j}$.
3. Find i and j such that $b_i = c_j$.
4. Return $a = i + \lfloor\sqrt{\ell}\rfloor \cdot j$.

Note that then $g^i = h \cdot g^{-\lfloor\sqrt{\ell}\rfloor \cdot j}$, so that in particular $g^{i+\lfloor\sqrt{\ell}\rfloor \cdot j} = h$. Also, every a can be written as $a = a_0 + a_1\lfloor\sqrt{\ell}\rfloor$ with $0 \leq a_0 \leq \lfloor\sqrt{\ell}\rfloor$ and $0 \leq a_1 \leq \lfloor\sqrt{\ell}\rfloor + 1$, so this algorithm will always return an answer. Baby step giant step takes $2\sqrt{\ell}$ multiplications and one inversion, which is already much better than $O(\ell)$!

Example. Let's use the baby step giant step algorithm to compute $\log_3(37)$ in $\mathbb{F}_{101}^*$. We have that $\langle 3 \rangle = \mathbb{F}_{101}^*$, so $\ell = |\mathbb{F}_{101}^*| = 100$, hence $\lfloor\sqrt{\ell}\rfloor = 10$. We first do the 'baby step', that is, we compute $b_i = 3^i$ for $0 \leq i \leq \lfloor\sqrt{\ell}\rfloor = 10$:

i	0	1	2	3	4	5	6	7	8	9	10
3^i	1	3	9	27	81	41	22	66	97	89	65

Now we do the 'giant step', this is, we compute $c_j = 37 \cdot 3^{-10j}$ for $j \geq 0$ until we find a value matching the table above:

j	0	1	2
c_j	37	13	81

We see from the tables that $b_4 = c_2$, that is, that $3^4 \equiv 37 \cdot 3^{-20} \bmod 101$, hence $a = 24$.

Baby step giant step gives us a large speed-up, but it uses $O(\sqrt{\ell})$ storage, which is huge! Another alternative is to use *Pollard's rho method*.

7 Pollard's rho method

Again, we assume that we have a subgroup $\langle g \rangle \subseteq \mathbb{F}_q^*$ of prime order ℓ, and that given $h \in \langle g \rangle$, we want to find $a = \log_g(h)$. In Pollard's rho method, we look for $b, c, b', c' \in \{1, \ldots, \ell\}$ such that

$$g^b h^c = g^{b'} h^{c'}.$$

This implies that in $\mathbb{F}_q^*$,

$$g^{b-b'} = g^{a(c'-c)},$$

hence

$$b - b' \equiv a(c' - c) \bmod \ell.$$

We find such b, c, b', c' by doing a pseudo-random walk on a graph with vertices G_i defined by: $G_0 = g$, $b_0 = 1$, $c_0 = 0$, and

$$(G_{i+1}, b_{i+1}, c_{i+1}) = \begin{cases} (G_i \cdot g, b_i + 1, c_i) & \text{if } G_i = 0 \bmod 3 \\ (G_i \cdot h, b_i, c_i + 1) & \text{if } G_i = 1 \bmod 3 \\ (G_i^2, 2b_i, 2c_i) & \text{if } G_i = 2 \bmod 3, \end{cases}$$

and aborting when we find a loop. Observe that finding a loop means that we have found some $i \neq j$ such that $G_i = G_j$, which gives that

$$g^{b_i} h^{c_i} \equiv G_i \equiv G_j \equiv g^{b_j} h^{c_j} \bmod \ell.$$

This algorithm loops after approximately $\sqrt{\frac{\pi}{2}\ell}$ steps and uses $O(1)$ storage, so is the most common algorithm in practise.

Example. Let's compute $a := \log_3(7)$ in $\mathbb{F}_{17}^*$ using Pollard's rho method. The algorithm outputs a list

$$(3,1,0),(9,2,0),(10,3,0),(2,3,1),(4,6,2),(11,6,3),(2,12,6).$$

The 4th and the 7th items in the list have the same first entry (i.e. $G_4 = G_7$), so we have a loop. Hence

$$3^3 \cdot 7^1 \equiv 3^{12} \cdot 7^6 \equiv 2 \bmod 17,$$

and $a \equiv (12-3)/(1-6) \bmod 16$, so

$$a = 11.$$

All the attacks presented so far do not make any use of any special properties of the group $\mathbb{F}_q^*$ or $\langle g \rangle$. Such algorithms are called 'generic algorithms' as they work for any group. For groups without special properties these algorithms are the best (known) to attack the discrete logarithm problem.