2WC12 Cryptography I – Fall 2014

October 16, 2014

Reminder about finite fields

For $\mathbb{F}_q$, $a \in \mathbb{F}_q$ we have

$$|\mathbb{F}_q| = q, \qquad |\mathbb{F}_q^*| = |\mathbb{F}_q \setminus \{0\}| = q - 1.$$

$\mathbb{F}_q$ is cyclic in respect to $+$:	$a \cdot \lvert\mathbb{F}_q\rvert = a \cdot q = 0$,	$a \cdot (q-1) = -a$,	$a \cdot (q+1) = a$.
$\mathbb{F}_q^*$ is cyclic in respect to $\cdot$:	$a^{\lvert\mathbb{F}_q^*\rvert} = a^{q-1} = 1$	$a^{q-2} = a^{-1}$,	$a^q = a$.

Diffie-Hellman key exchange

Alice and Bob want to secretly exchange a shared key.
First they agree on a finite field $\mathbb{F}_q$ (e.g., $\mathbb{F}_{p^n}$ represented as $\mathbb{F}_p[x]/_{f(x)\mathbb{F}_p[x]}$ with an irreducible polynomial f of degree n) and a generator g of that field with $\mathbb{F}_q^* = \langle g \rangle$. These parameters are public. (Actually any cyclic group works for DH.)
Now, the Diffie-Hellman key exchange works as follows:

1. Alice picks a random $a \in \mathbb{F}_q^*$ and sends $a' = g^a \in \mathbb{F}_q$ to Bob.
2. Bob picks a random $b \in \mathbb{F}_q^*$ and sends $b' = g^b \in \mathbb{F}_q$ to Alice.
3. Alice computes $b'^a = g^{b^a} = g^{ab} \in \mathbb{F}_q$.
4. Bob computes $a'^b = g^{a^b} = g^{ab} \in \mathbb{F}_q$.

Now Alice and Bob share the secret value $g^{ab} \in \mathbb{F}_q$ which they can use to compute a secret key for private communication, e.g., using AES.

An attacker only sees g^a and g^b on the wire and has to compute the *discrete logarithm* in $\mathbb{F}_q$. This is believed to be difficult (for large q, a, and b).

Problem: authentication, man in the middle, ...

ElGamal encryption

Alice choses a finite field $\mathbb{F}_q$, a generator g, and a random $a \in \mathbb{F}_q^*$ and computes $h = g^a$. The public key of Alice is $(\mathbb{F}_q, g, h)$; her secret private key is a.

Encryption: Bob wants to send a message $m \in \mathbb{F}_q$ to Alice:

- Bob picks a random $b \in \{1, \ldots, q-1\}$ and computes $c_1 = g^b$.
- Bob computes the shared secret $s = h^b = g^{a^b} = g^{ab}$.
- Bob computes $c_2 = m \cdot s$.
- Bob sends the ciphertext $(c_1, c_2) = (g^b, m \cdot g^{ab})$ to Alice.

Observe that it is easy to compute g^{ab} when m is known! Thus, a new b is used for each message.

Decryption: Alice decrypts a ciphertext c_1, c_2 with her private key a.

- Alice computes the shared secret $s = c_1^a = g^{b^a} = g^{ab}$.
- Alice computes $c_2 \cdot s^{-1} = m \cdot g^{ab} \cdot (g^{ab})^{-1} = m$.

Observe that s^{-1} can be computed directly as $s^{-1} = c_1^{q-a-1} = g^{b^{q-a-1}} = g^{b(q-a-1)}$ because

$$s \cdot s^{-1} = g^{ab} \cdot g^{b(q-a-1)} = g^{ab+b(q-a-1)} = g^{b(a+q-a-1)} = g^{b(q-1)} = \left(g^{q-1}\right)^b = 1^b = 1.$$

ElGamal signatures

Use a hash function $\mathrm{H} : \{0,1\}^* \longmapsto \mathbb{Z}$.
Chose $\mathbb{F}_q$, a generator g, and a private $a \in \{1, \ldots, q-1\}$ and compute the public key $h = g^a$.

Sign: Pick a random nonce k. Compute $r = g^k$ and $s \equiv k^{-1}\left(\mathrm{H}(m) - ar\right) \mod (q-1)$.
The signature is the pair (r, s),

Verify: Check if $g^{\mathrm{H}(m)} = h^r \cdot r^s$:

$$h^r \cdot r^s = (g^a)^r \cdot (g^k)^s = g^{ar} \cdot g^{k \cdot k^{-1}(\mathrm{H}(m) - ar)} = g^{\mathrm{H}(m)}$$

Pitfalls with Signatures

If for some message m with signature (r, s) the random value k becomes known, one can compute

$$a \equiv \frac{\mathrm{H}(m) - k \cdot s}{r} \mod (q-1)$$

and get the secret key.
If the same k is used for more than one message, i.e., $s_i \equiv k^{-1}(\mathrm{H}(m_i) - ar)$ take

$$s_1 - s_2 \equiv k^{-1}(\mathrm{H}(m_1) - ra - \mathrm{H}(m_2) + ra) \mod (q-1)$$

to compute $k \equiv \frac{\mathrm{H}(m_1) - \mathrm{H}(m_2)}{s_1 - s_2} \mod (q-1)$ and then retrieve a as above.

Do not reuse k!

k is often called a *nonce*, a "number used only once". → PS3 hack
Other solution: make k deterministically depend on m and a. ⇒ RFC 6979

Are there other ways to break the signature scheme?
Signature depends only on $\mathrm{H}(m)$, not on m itself, so take a signature (r, s) on m and find $m' \neq m$ with $\mathrm{H}(m') = \mathrm{H}(m)$. Then (r, s) is also a signature on m'.
Cryptographic hash functions are expected to be resistant against finding second preimages (and also against finding preimages in general).

A more active attacker could generate 2 messages $m \neq m'$ with $\mathrm{H}(m) = \mathrm{H}(m')$ one of which he asks Alice to sign. The signature is valid for both m and m'. Requiring the hash function to avoid this is a stronger requirement, called collision resistance (the attacker can vary both messages).
Cryptographic hash functions are expected to be collision resistant.

Other attacks?
Find a, given $h = g^a$, i.e., solve the DLP.

Example. $\mathbb{F}_7^* = \langle 3 \rangle$, find $a = \log_3 5$, i.e., the integer $1 \leq a \leq 6$ with $3^a = 5$.
For this problem size, we can simply try $3^2 = 2$, $3^3 = 6$, $3^4 = 4$, $3^5 = 5 \quad \Rightarrow \quad a = 5$.

Example. $\mathbb{F}_{37}^* = \langle 2 \rangle$, find $a = \log_2 17$.
$|\mathbb{F}_{37}^*| = 36 = 2^2 \cdot 3^2$; thus $\mathbb{F}_{37}^*$ has subgroups of order 2 ($\{2^0, 2^{18}\} = \{1, 36\}$), 3 ($\{2^0, 2^{12}, 2^{24}\} = \{1, 26, 10\}$), 4, 6, 9, 12, and 18. Solving the DLP in the subgroup of order 2 (i.e., mod 2) has 2 choices and gives the LSB of a.
Write $a = a_0 + 2a_1$, then $2^a = 17$ gives

$$(2^a)^{18} = 2^{a_0 \cdot 18 + a_1 \cdot 2 \cdot 36} = 2^{a_0 \cdot 18} = 17^{18} = 36$$

$$\Rightarrow a_0 = 1;\ a_1 = ?$$

Try the same modulo 3, write $a = b_0 + 3b_1$:

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

$$\Rightarrow b_0 \neq 0, \text{ test for } b_0 = 1 \text{ or } b_0 = 2;\ 2^{12} = 26 \quad \Rightarrow b_0 = 1$$

To get more information on a, consider $a = 1 + 2a_1$ again. This implies

$$17 = 2^1 \cdot 2^{2a_1} \quad | \cdot 2^{-1} \qquad \Rightarrow \qquad 27 = 2^{2a_1}$$

Write $a_1 = c_0 + 2c_1$

$$2^{2a_1 \cdot 9} = 2^{18c_0 + 36c_1} = 2^{18c_0} = 27^9 = 36 \qquad \Rightarrow c_0 = 1$$

$$a = a_0 + 2(c_0 + 2c_1) \qquad \Rightarrow a \equiv 1 + 2 \cdot 1 \mod 4$$

Same for $b_0 + 3b_1 = b_0 + 3(d_0 + 3d_1) = b_0 + 3d_0 + 9d_1 = 1 + 3d_0 + 9d_1$:

$$2^{a \cdot 4} \cdot 2^{-1} = 2^{12d_0 + 36d_1} = 27^4 = 10,$$

since $2^{24} = 10$ we get $d_0 = 2$ and $a \equiv 1 + 2 \cdot 3 \mod 9$.
In total we have

$$a \equiv 3 \mod 4$$
$$a \equiv 7 \mod 9$$

Solve this system of linear congruences by CRT to get

$$a \equiv 7 \mod 36, \text{ thus } 2^7 = 17.$$

This example shows that to solve the DLP in $\mathbb{F}_q^*$ it suffices to solve it in every subgroup of prime order (potentially multiple times to deal with prime powers) and then to combine the results from the small DLP computations using CRT. This method is known as *Pohlig-Hellman* attack:

Pohlig-Hellman attack

Let $G = \langle g \rangle$ and let $\text{ord}(G) = p_1^{l_1} \cdot \cdots \cdot p_n^{l_n}$ with $p_i \neq p_j$ and $l_i \geq 1$ for $1 \leq i, j \leq n$ and $i \neq j$. Determine $\log_g h = a$ by computing $a \bmod p_i^{l_i}$ as follows for all i and then using CRT to combine the results:
Write $a = a_{i_0} + a_{i_1} p_i + a_{i_2} p_i^2 + \cdots + a_{a_{l_i}} p_i^{l_i}$.

Compute $g_i := g^{\text{ord}(G)/p_i}$.
Compute $a_{i_0} := \log_{g_i} h^{\text{ord}(G)/p_i}$ (this is a DL computation in group of size p_i)
Set $h_i := h$
For $j := 1$ to $l_i - 1$
    compute $h_i := h_i / g^{a_{i_{j-1}} p_i^{j-1}}$

compute $a_{i_j} := \log_{g_i} h_i^{\mathrm{ord}(G)/p_i^j}$

Return $a \bmod p_i^{l_i}$

To protect against the Pohlig-Hellman attack choose groups having a large prime order subgroup — so clearly not $\mathbb{F}_{2^n+1}^*$ (provided 2^n+1 is a prime power). Some protocols work in a large prime order subgroup, which can give speed advantages over working in the full group.

Example. DSA, the Digital Signature Algorithm.
Let $\langle g \rangle \subseteq \mathbb{F}_q^*$ with $\mathrm{ord}(\langle g \rangle) = l$, some prime. Public key $h = g^a$, secret key a.
The signature on message m is (r, s), computed as follows:

- pick random nonce k, compute (in $\mathbb{F}_q^*$) $r' = g^k$, embed r' into $\mathbb{Z}$ and compute $r \equiv r' \mod l$,
- compute $s \equiv k^{-1}(\mathrm{H}(m) + ar) \mod l$.

To verify this signature compute $w \equiv s^{-1} \mod l$, $u_1 \equiv \mathrm{H}(m) \cdot w \mod l$, $u_2 \equiv r \cdot w \mod l$, and $v' = g^{u_1} \cdot h^{u_2}$. Check whether the embedding of v' into $\mathbb{Z}$ satisfies $v \equiv r \mod l$.
A properly formed signature passes this test because

$$v' = g^{u_1} \cdot h^{u_2} = g^{\mathrm{H}(m) \cdot w} \cdot g^{a(rw)} = g^{w(\mathrm{H}(m)+ar)} = g^k = r'.$$

(Note that the order of $\langle g \rangle$ is l, so the computations mod l make sense in the exponent.)

Now, how hard is it to solve DLPs anyway?
One can find out in at most $\mathrm{ord}(\langle g \rangle)$ trials (or rather in $c \cdot l$, for some c and l the largest prime factor of $\mathrm{ord}(\langle g \rangle)$) but this is not the fastest possibility. With 50% probability, the DLP is solved after half the trials.

Let $\log_g h = a$ and write $a = a_0 + a_1 \lfloor \sqrt{l} \rfloor$ with $0 \le a_0 < \lfloor \sqrt{l} \rfloor$ and $0 \le a_1 \le \lceil \sqrt{l} \rceil$ (where $\lfloor \ldots \rfloor$ denotes rounding down and $\lceil \ldots \rceil$ denotes rounding up).
Thus $g^a = g^{a_0 + a_1 \lfloor \sqrt{l} \rfloor} = g^{a_0} \cdot g^{a_1 \lfloor \sqrt{l} \rfloor} = h$, and $h \cdot g^{-a_1 \lfloor \sqrt{l} \rfloor} = g^{a_0}$.
Then computing

$$b_i := g^i$$

for $0 \le i \le \lfloor \sqrt{l} \rfloor$ takes $\lfloor \sqrt{l} \rfloor$ multiplications, computing

$$c_j := h \cdot g^{-\lfloor \sqrt{l} \rfloor \cdot j}$$

for $0 \le j \le \lceil \sqrt{l} \rceil$ takes another $\lceil \sqrt{l} \rceil$ operations. Now, find i, j such that $b_i = c_j$, i.e., $g^i = h \cdot g^{-\lfloor \sqrt{l} \rfloor \cdot j}$. This procedure will find a_0 and a_1 in at most $2\sqrt{l}$ multiplications and one inversion — this is much faster than l multiplications.

This algorithm is called the *Baby-Step Giant-Step* (BSGS) algorithm.

Example. Find a for $3^a = 37$ in $\mathbb{F}_{101}$, i.e., $g = 3$ and $h = 37$.
$l = 100$, thus $\lfloor \sqrt{l} \rfloor = 10$.
Compute $b_i := g^i = 3^i$ for $0 \le i \le 10$:

baby step 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, compute $c_j := h \cdot g^{-\lfloor \sqrt{l} \rfloor \cdot j} = 37 \cdot 3^{-10 \cdot j}$ for $0 \le j \le 10$:

giant step j	0	1	2	3	4	5	6	7	8	9	10
$37 \cdot 3^{-10 \cdot j}$	37	13	81	...							

Thus $a_0 = 4$ and $a_1 = 2$, $\quad \Rightarrow \quad a = a_0 + 10a_2 = 4 + 10 \cdot 2 = 24$.

To solve the DLP in a group of size l it takes $O(\sqrt{l})$ operations; to make use of smaller size subgroups, combine this algorithm with the Pohlig-Hellman attack. BSGS is good in speed but quickly grows problematic in storage size — the b_i have to be stored and sorted, that is $\sqrt{l}$ elements.

Pollard's rho method drastically reduces the storage, at the expense of randomizing the algorithm. In order to find a such that $g^a = h$, the basic idea is to find integers b, c, b', c' such that

$$g^b h^c = g^{b'} h^{c'} \Rightarrow g^{b-b'} = h^{c'-c} = (g^a)^{c'-c} = g^{a(c'-c)} \Rightarrow b - b' \equiv a(c' - c) \mod l.$$

$\rightarrow$ Slides.
Define the pseudo-random walk iteratively, e.g., as follows:
$G_0 = g, \quad b_0 = 1, \quad c_0 = 0,$

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

This results in a loop after approximately $\sqrt{\frac{\pi}{2} l}$ steps.
Improve the algorithm by using Floyd's cycle-finding algorithm, i.e., maintain to values $G' (\cong G_i)$ and $G'' (\cong G_{2i})$ and step twice on G'' for each step on G'; halt when $G' = G''$.

The attacks presented so far do not make use of any special properties of the group. Such algorithms are called "generic algorithms", they work for any group. For groups without special properties these algorithms are the best to attack the DLP. Given that BSGS and Pollard's rho method both take $O(\sqrt{l})$ steps in a group of size l and that Pollard's rho method takes $O(1)$ storage, all attacks use Pollard's rho method in practice, sometimes with some small speedups depending on the group.