Discrete logarithm and related cryptosystems - II

Diego F. Aranha

Institute of Computing
UNICAMP

Security issues on Diffie-Hellman related problems

Prime order groups

Uniformity

Let $\mathbb{G}$ be a group of prime order q with $|q| = n$ and generator g. If x_1, x_2 are randomly chosen from $\mathbb{Z}_q$, there is a negligible function δ, such that for all $z \in \mathbb{G}$, we have

$$\Pr[(g^{x_1 x_2}) = z] = \frac{1}{q} - \delta(n).$$

First guess: When $\mathbb{G} = \mathbb{Z}_p^*$, p prime, we have that $q = p - 1$ and thus $\mathbb{G}$ does not have prime order.

A better one: Use the quadratic residue subgroup $\mathbb{QR}_p$ of $\mathbb{Z}_p^*$. If $p = 2q + 1$, with q prime, $\mathbb{QR}_p$ has prime order $q = \frac{(p-1)}{2}$.

Example: $(\mathbb{QR}_{11}, \times \bmod 11)$

a	$a^2 \bmod 11$
1	1
2	4
3	9
4	5
5	3
6	3
7	5
8	9
9	4
10	1

So, $\mathbb{QR}_{11} = \{1, 3, 4, 5, 9\}$ has order $\frac{(11-1)}{2} = 5$.

Exercise: Verify that $\mathbb{QR}_p$ is, indeed, a group.

Security Issues on ElGamal encryption

Remember $\mathbb{Z}_p^*$-based ElGamal encryption and decryption:

Encryption of m:

1. Choose integer k uniformly random in $\mathbb{Z}_{p-1}$.
2. Compute $Enc_K(m, k) = (c_1, c_2)$, where $c_1 = \alpha^k \bmod p$, $c_2 = m\beta^k \bmod p$, and $\beta \equiv \alpha^a \pmod p$.

Decryption: Compute $Dec_K(c_1, c_2) = c_2(c_1^a)^{-1} \bmod p$.

Security Issues on ElGamal encryption

Remember $\mathbb{Z}_p^*$-based ElGamal encryption and decryption:

Encryption of m:

1. Choose integer k uniformly random in $\mathbb{Z}_{p-1}$.
2. Compute $Enc_K(m, k) = (c_1, c_2)$, where $c_1 = \alpha^k \bmod p$, $c_2 = m\beta^k \bmod p$, and $\beta \equiv \alpha^a \pmod p$.

Decryption: Compute $Dec_K(c_1, c_2) = c_2(c_1^a)^{-1} \bmod p$.

Now note that:

- Ciphertexts (c_1, c_2) and $(c_1, c_2' = c_2 \cdot m')$, for any $m' \in \mathbb{Z}_p^*$, are both valid. This may be used by an adversary that intercepted (c_1, c_2), e.g. in a chosen plaintext attack, or in an impersonation attack.
- Furthermore, to avoid the detection of ciphertexts with same c_1, the adversary can multiply c_1 and c_2' by α^y and β^y, respectively, where y is a random element in $\mathbb{Z}_{p-1}$.
- Verify these claims!

Implementation issues on discrete log-based encryption

Encoding a message as a group element depends on the group representation.

Sharing system parameters among different users does not appear to have security implications.

Avoid generating your parameters and use standardized ones!

Computing discrete logarithms

There are many approaches for computing discrete logarithms. Some notables ones are:

- Shanks' (Babystep-Giantstep) Algorithm
- Pollard ρ Method
- Pohlig-Hellman Algorithm
- Index Calculus Method

Important: It is common for advances in integer factoring to produce advances in discrete logarithms over $\mathbb{Z}$.

Shanks' (Babystep-Giantstep) Algorithm

Shanks' Algorithm

Input: group $\mathbb{G}, n = |\mathbb{G}|, \alpha, \beta$
Output: $\log_\alpha \beta$

1. $m \leftarrow \lceil \sqrt{n} \rceil$
2. **For** $j \leftarrow 0$ **to** $m-1$ compute α^{mj}
3. Store the pairs (j, α^{mj}) in a hash table L_1, keyed by the second component
4. **For** $i \leftarrow 0$ **to** $m-1$ compute $\beta\alpha^{-i}$
5. Store the pairs $(i, \beta\alpha^{-i})$ in a hash table L_2, keyed by the second component
6. Find pairs $(j, y) \in L_1$ and $(i, y) \in L_2$
7. **Return** $\log_\alpha \beta = (mj + i) \bmod n$

See Example 2.23 in "Introduction to Mathematical Cryptography."

Shanks' Algorithm

Correctness:

$$\alpha^{mj} = y = \beta\alpha^{-i} \Rightarrow \alpha^{mj+i} = \beta.$$

Complexity:

- Computing tables L_1, L_2 takes $O(m) = O(n^{1/2}) = O(2^{k/2})$ steps, where $k = O(\log_2 n)$ is the size (number of bits) of the input n.
- Finding pairs $(j, y) \in L_1$ and $(i, y) \in L_2$ also takes $O(m)$ steps, since the tables are ordered (keyed) by these values.
- Thus, Shanks' Algorithm is exponential in time and space. However, it is much better than an $O(n)$-time algorithm for the exhaustive search of the discrete log.

Note that:

- The number of steps to perform one multiplication of two group elements is usually very small when compared to $O(n^{1/2})$, so we omit it from the discussion above.

Pollard ρ Method

Pollard's Algorithm has the same time complexity as Shanks' but with constant ($O(1)$) space complexity, a big improvement! Let's first discuss Pollard's method in a more abstract setting.

Suppose we have a finite set S and a random map $f : S \to S$. Then, the sequence

$$x_0 = x,\ x_1 = f(x_0),\ x_2 = f(x_1)\ \ldots$$

will eventually produce $x_T = x_{M+T}$.

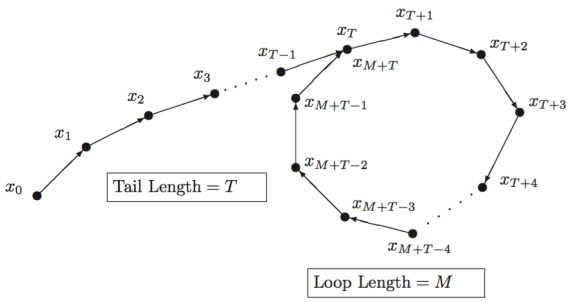

Figure 4.1: Pollard's ρ method

Figure: From Intro. to Math. Crypto. - Hoffstein et al.

Pollard ρ Method

Question: Can we find collisions without storing $x_0, x_1, \ldots, x_T, \ldots x_{T+M}$?

Pollard's answer: Yes! Just calculate simultaneously the two sequences

$$x_0, x_1, x_2, x_3 \ldots \quad \text{and} \quad x_0, x_2, x_4, x_6, \ldots$$

Then,

- $x_{2i} = x_i$ iff $(i \leq T$ and $2i \equiv i \pmod{M})$ iff $M|i$;
- that is, we get $x_{2i} = x_i$ when i is equal to the first multiple of M;
- and since one of M in $T, T+1, \ldots T+M-1$ is divisible by M, we have that

$$x_{2i} = x_i \text{ for some } 1 \leq i < T + M.$$

Now, Theorem 4.47 of the IMC book proves that, if the map f is sufficiently random, then $T + M$ is, in average, $\approx 1.25 \cdot \sqrt{|S|}$.

Thus, we can find collisions in constant space and $O(\sqrt{|S|})$ time.

Pollard ρ Method

We now show how to instantiate a suitable map f in order to find $\log_\alpha \beta$ in a group $\mathbb{G}$, where $\alpha \in \mathbb{G}$ has order n.

First define $S_1 \cup S_2 \cup S_3$, a partition of $\mathbb{G}$ in three different sets of similar size.

Now define the map f as

$$f(x, a, b) = \begin{cases} (\beta x, a, b+1), & \text{if } x \in S_1; \\ (x^2, 2a, 2b), & \text{if } x \in S_2; \\ (\alpha x, a+1, b), & \text{if } x \in S_3. \end{cases}$$

Note that map f merely produces triples of the type $(\alpha^A \beta^B, A, B)$.

Pollard ρ Method

Now, starting with triple $(1, 0, 0)$, we iterate f and eventually find two triples (x_{2i}, a_{2i}, b_{2i}) and (x_i, a_i, b_i) with $x_{2i} = x_i$.

Hence:

$$\begin{aligned}
\alpha^{a_{2i}}\beta^{b_{2i}} &= \alpha^{a_i}\beta^{b_i} \\
\alpha^{a_{2i}+(\log_\alpha \beta)b_{2i}} &= \alpha^{a_i+(\log_\alpha \beta)b_i} \\
a_{2i} + (\log_\alpha \beta)b_{2i} &\equiv a_i + (\log_\alpha \beta)b_i \pmod{n} \\
\log_\alpha \beta &= \frac{a_i - a_{2i}}{b_{2i} - b_i} \bmod n
\end{aligned}$$

Pollard ρ Method

Pollard ρ Algorithm

Input: Group $\mathbb{G}$, elements α, β, such that $\beta \in \langle\alpha\rangle$.

1. Define the partition $S_1 \cup S_2 \cup S_3$ for $\mathbb{G}$, used in map f.
2. $(x, a, b) \leftarrow f(1, 0, 0)$
3. $(x', a', b') \leftarrow f(x, a, b)$
4. **While** $x \neq x'$
   - $(x, a, b) \leftarrow f(x, a, b)$
   - $(x', a', b') \leftarrow f(x', a', b')$
   - $(x', a', b') \leftarrow f(x', a', b')$
5. **If** $\gcd(b' - b, n) \neq 1$ **return** FAIL;
   otherwise, **return** $(a - a')(b' - b)^{-1} \bmod n$.

Important: As is evident, this algorithm may not return a valid result. This is, however, unlikely for carefully chosen maps f.