Cryptography 101: Discrete Log, Diffie-Hellman and ElGamal

ComComX • 16 Apr 2026

In which we show that how secure communication is possible without any prearrangement or any shared randomness.

Preliminaries

- $\mathbb{Z}_p$: ring of integers modulo p, which is defined over $\{0, \dots, p-1\}$ with addition and multiplication modulo p
- $\mathbb{Z}_p^*$: multiplicative group defined by $\mathbb{Z}_p$, which is defined over $\{a \in \mathbb{Z}_p | \gcd(a, p) = 1\}$ with multiplication modulo p.
- PRIMES_λ: the set of all λ-bit primes, i.e., all primes in $[2^{\lambda-1}, 2^\lambda]$. By the *Prime number theorem*, we know that there is $\Omega(1/\lambda)$ fraction of λ-bit numbers are primes.
- $\mathsf{Gen}(G)$: set of all generators of a cyclic group G. g is a generator of the group G if $G = \{g^x | x \in \mathbb{Z}\}$.

From now, we assume p is a λ-bit prime. As the consequence, $\mathbb{Z}_p^*$ is a cyclic group over $\mathbb{Z}_p \backslash \{0\}$ with $|\mathsf{Gen}(\mathbb{Z}_p^*)| = \Omega\left(\frac{p}{\log \log p}\right)$ generators.

We have the hardness of some arithmetic operations in $\mathbb{Z}_p^*$:

- **Multiplication**: $O(\log^2 p)$ using naive algorithms, $O(\log p \log \log p)$ using FFT algorithm.
- **Exponentiation**: $O(\log^3 p)$ using repeated squaring method ($O(M \log b)$ time to compute a^b where M is the time for multiplication).
- **Inverse**: same as exponentiation, which is $O(\log^3 p)$. This is because $a^{-1} = a^{-1}a^{p-1} \pmod p = a^{p-2} \pmod p$ by the *Fermat's little theorem*.

Discrete logarithm problem

For a generator g and $h \in G$, there must be an x such that $h = g^x$. This number is called the discrete logarithm of h w.r.t. the generator g, denoted by $x = \mathrm{dlog}_g(h)$.

Definition 1 (Discrete logarithm (DL) problem)**.** *Given h and g, the problem is to compute $x = \mathrm{dlog}_g(h)$.*

Unlike the three operations mentioned previously, discrete log is believed to be hard: trivial bruteforce requires time $O(p \cdot \mathrm{poly}(\log p))$, which is exponential in λ. All existing algorithms, either for general groups (like Baby-step Giant-step and Pohlig–Hellman algorithms) or for specific $\mathbb{Z}_p^*$ (like Index-Calculus and Number field sieve algorithms) require time exponential in λ. Therefore, it is believed that the problem is hard to solve:

Assumption 2 (Hardness of discrete-log)**.** *For any PPT algorithm A,*

$$\Pr_{p \leftarrow \mathsf{PRIMES}, g \leftarrow \mathsf{Gen}(\mathbb{Z}_p^*), x \leftarrow \mathbb{Z}_{p-1}}[A(p, g, g^x) = x] = \mathrm{negl}(\lambda).$$

Here, our assumption is that the discrete-log problem is not just hard but even **hard on average**, as all p, g, x are random.

Note: asymmetric cryptography we are using today relies on the assumption regarding the hardness of two computational problems: one is the above Discrete-log (used in Diffie-Hellman, DSA/ECDSA, and Elliptic Curve Cryptography) and the another is the Integer Factorization (used in the RSA algorithm). This seems to be... embarrassing!?! The quote below is from the introduction of *Foundations of Cryptography – A Primer* by Oded Goldreich:

> *"The way to demonstrate that a definition is viable (and that the corresponding intuitive security concern can be satisfied at all) is to construct a solution based on a better understood assumption (i.e., one that is more common and widely believed). [...] Thus, facts that were not known to hold at all (and even believed to be false), were shown to hold by reduction to widely believed assumptions (without which most of modern cryptography collapses anyhow). To summarize, not all assumptions are equal, and so reducing a complex, new and doubtful assumption to a widely-believed simple (or even merely simpler) assumption is of great value."*

More note: in 1994, Peter Shor discoverd that quantum computer can solve Discrete-log and Integer Factorization in polynomial time (FOCS'94). The algorithm is now named after him, the Shor's algorithm. Before that, quantum

computing was essentially a physics toy or a theoretical concept discussed by people like Richard Feynman. Shor's algorithm turned this scientific curiosity into a national security priority.

Constructing OWF from Discrete-log: the simplest way to constructing a OWF under the assumption about hardness of discrete-log is:

$$f_\lambda(p, g, x) = (p, g, g^x)$$

where p is a λ-bit prime, g is a generator of $\mathbb{Z}_p^*$, and $x \in \mathbb{Z}_{p-1}$. Here, no adversary can recover (p, g, x) given (p, g, g^x) with noticeable advantage since finding x given g^x is assumed to be hard.

Diffie-Hellman (DH) Problem

Let $\mathsf{GroupGen}$ be a PPT algorithm that, given an input 1^λ, outputs a description of a cyclic group G, which includes a prime G_λ and a generator g.

Definition 3 (Computational diffie-Hellman (CDH) problem)**.** *Given a cyclic group G of order Q, a generator g, and two elements g^x and g^y of $\mathbb{Z}_Q$. The problem is to compute g^{xy}.*

If we can solve the DL efficiently then clearly we can also solve CDH efficiently. On the other hand, if DL is hard, then it is, again, believed that CDH is also hard:

Assumption 4 (Hardness of Computational Diffie-Hellman (CDH))**.** *For any PPT algrithm A,*

$$\Pr_{(G,\, g)\leftarrow\mathsf{GroupGen}(1^\lambda)\,(x,y)\leftarrow\mathbb{Z}_G}[A(G_\lambda, g, g^x, g^y) = g^{xy}] = \mathrm{negl}(\lambda).$$

We often use the following stronger assumption:

Assumption 5 (Hardness of Decisional Diffie-Hellman (DDH))**.** *For any PPT algrithm A,*

$$\begin{aligned}&\Pr_{(G_\lambda,g)\leftarrow\mathsf{GroupGen}(1^\lambda),(x,y)\leftarrow\mathbb{Z}_{G_\lambda}}[A(G_\lambda, g, g^x, g^y, g^{xy}) = 1]\\ &\quad - \Pr_{(G_\lambda,g)\leftarrow\mathsf{GroupGen}(1^\lambda),(x,y,z)\leftarrow\mathbb{Z}_{G_\lambda}}[A(G_\lambda, g, g^x, g^y, g^z) = 1]\\ &\qquad = \mathrm{negl}(\lambda).\end{aligned}$$

Diffie-Hellman key exchange

The protocol is as follows:

1. $(G_\lambda, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$. This group description is publicly known
2. Alice samples $x \leftarrow \mathbb{Z}_{G_\lambda}$ and sends g^x to Bob
3. Simultaneously, Bob samples $y \leftarrow \mathbb{Z}_{G_\lambda}$ and sends g^y to Alice

4. Alice computes $g^{xy} = (g^y)^x$, Bob computes $g^{xy} = (g^x)^y$

Clearly, Alice and Bob both have g^{xy} as the shared random string at the end of the protocol. What the adversary can see over the channel is (G_λ, g, g^x, g^y). Then,

- If we assume CDH: the adversary can not compute the string g^{xy} in polynomial time. Alice can use a hardcore predicate for the OWF
$$f_\lambda(G_\lambda, g, g^x, g^y, g^{xy}) = (g_\lambda, g, g^x, g^y)$$
to pad the secret message to Bob.
- If we assume DDH: the adversary can not distinguish g^{xy} from a truly random string, so Alice can use g^{xy} to pad the secret message to Bob.

Note: the secret message in the above context is the key need to be exchanged (?). Additionally, this protocol is vulnerable to a **man-in-the-middle attack**.

ElGamal Encryption

The Elgamal encryption scheme is a public-key encryption scheme based on DDH. The scheme is as follows:

- **Key generation** $\mathsf{Gen}(1^\lambda)$:
    1. Generate group G: $(G_\lambda, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$
    2. $x \leftarrow \mathbb{Z}_{|G|}$
    3. Output public key $pk = (G_\lambda, g, g^x)$ and secret key $sk = x$
- **Encryption** $\mathsf{Enc}(1^\lambda, pk, m \in G)$:
    1. $y \leftarrow \mathbb{Z}_{|G|}$
    2. Output $c = (g^y, (g^x)^y m)$
- **Decryption** $\mathsf{Dec}(1^\lambda, sk, c)$:
    1. Extract x from sk
    2. Extract g^y and $g^{xy}m$ from c
    3. Compute $g^{xy} = (g^y)^x$
    4. Output $(g^{xy})^{-1} g^{xy} m$.

The correctness of the scheme is immediate. The following theorem shows the security of the scheme:

Theorem 6 (Security of Elgamal encryption)**.** *Under the DDH assumption, the Elgamal encryption scheme is CPA-secure.*

Proof sketch. If A, by contradiction, wins the IND-CPA game, we use A to construct a PPT algorithm that solves the DDH problem with successful probability noticeably better than 1/2. Particularly, D receives as input a challenge (G_λ, g^x, g^y, Z) and is asked to conclude whether Z is g^{xy} or g^z for a random z. Let D simulate A and when A sends two messages m_0, m_1:

- D samples a bit b randomly
- D sends $(g^y, Z \cdot m_b)$ to A
- D outputs 1 if A wins the game, otherwise outputs 0.

Observe that if $Z = g^{xy}$ then A is playing exactly an instance of IND-CPA. Otherwise, A has no clue what is b since $g^z \cdot m_b$ is truly random to it. □