Diffie-Hellman Key Exchange

First published public-key algorithm

- Practical method for public exchange of a secret key
- Used in a number of commercial products

Idea:

- Enables two users to securely exchange a secret or symmetric key for subsequent encryption (Ex. AES)
- Value of the secret key depends on the participants (and their private and public key information)
- Security relies on:
  - Exponentiation in a finite (Galois) field (modulo a prime or a polynomial) is easy
  - Computing discrete logarithms (similar to prime factoring) is hard

Diffie-Hellman Setup

All uses agree on global parameters:

- A large prime integer: q
- A primitive root of q: a

Suppose users A and B wish to exchange a secret key

- User A: selects a random integer $X_a < q$
  - Computes $X_A = a^{x_A} \bmod q$
- User B: selects a random integer $X_b < q$
  - Computes $X_B = a^{x_B} \bmod q$

Each side keeps X as a private key and Y as the private key

K_{AB} is the exchanged secret key for users A and B:

- $K_{AB} = {Y_B}^{X_A} \bmod q$ (**User A calculation**)

$$
\begin{aligned}
&= (a^{X_B} \bmod q)^{X_A} \bmod q \\
&= (a^{X_B}) \bmod q \\
&= a^{X_B X_A} \bmod q \\
&= (a^{X_A})^{X_B} \bmod q \\
&= (a^{X_A} \bmod q)^{X_B} \bmod q \\
&= {Y_A}^{X_B} \bmod q \text{ (\textbf{User B calculation})}
\end{aligned}
$$

A and B subsequently use K_{AB} for symmetric encryption

Attackers knows q, a, Y_A, Y_B; They need to know X_A or X_B

- $X_A = \mathrm{dlog}_{a,q}\, Y_A$, $X_B = \mathrm{dlog}_{a,q}\, Y_B$,
- This is hard for large numbers

Diffie-Hellman Example

Problem: Alice and Bob want to exchange secret keys

- Agree on prime q = 353, a = 3
- Select random private keys:
  - Alice chooses $X_A = 97$
  - Bob chooses $X_B = 233$
- Compute respective public keys:
  - $Y_A = 3^{97} \bmod 353 = 40$ (Alice)
  - $Y_B = 3^{233} \bmod 353 = 248$ (Bob)
- Compute shared session key as:
  - $K_{AB} = {Y_B}^{X_A} \bmod 353 = 248^{97} = 160$ (Alice)
  - $K_{AB} = {Y_A}^{X_B} \bmod 353 = 40^{233} = 160$ (Bob)

Man-in-the-Middle Attack

Key exchange protocols:

- Public keys could be between two users A and B
- Could be between a group of users
- Both are vulnerable to a Man-in-the-middle attack

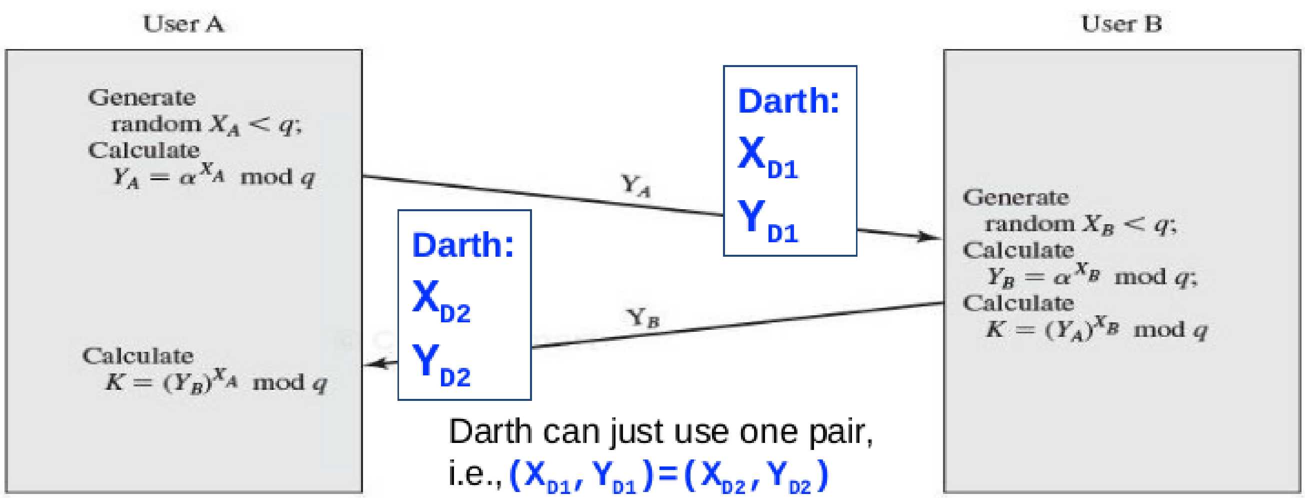

Figure 1: Diagram of a man-in-the-middle attack

$K_{AD} = {Y_{D2}}^{X_A} \bmod q = {Y_A}^{X_{D2}} \bmod q$

$K_{BD} = {Y_B}^{X_{D1}} \bmod q = {Y_{D1}}^{X_B} \bmod q$

Darth can eavesdrop or modify messages

Due to the authenticity of two parties not being established

Use public-key certificate and digital signatures to overcome this

Ephemeral Diffie-Hellman and Perfect Forward Secrecy (PFS)

Fixed Diffie-Hellman: embeds the server's public parameter in the certificate, and the CA then signs the certificate. The certificate contains the Diffie-Hellman public-key parameters that never change. (Prone to man-in-the-middle attack)

Ephemeral Diffie-Hellman: uses temporary, public keys. Each instance or run of the protocol uses a different public key. Each instance or run of the protocol uses a different public key. The authenticity of the server's temporary key can be verrified by checking the signature on the key. Because the public keys are temporary, a compromise of the server's long term signing key does not jeopardize the privacy of past sessions. This is known as perfect forward secrecy.

Only 44% of connects use ephemeral Diffie-Hellman

ElGamal Cryptography

Uses concepts from Diffie-Hellman and further add security

- Does this by using finite (Galois) field
- Security depends on difficulty of computing discrete logarithms (Like Diffie-Hellman)
- Used in a number of standards
  - Digital signature standard (DSS)
  - Email standard (S/MIME)

ElGamal Setup

All users agree on global parameters:

- A large prime integer: q
- A primitive root of q: a

Users B wants to securely send a message to user A

User A

- Selects a random integer $X_A < q - 1$
- Computes $Y_A = a^{X_A} \bmod q$
- A's private key is X_A; A's public key is $\{q, a, Y_A\}$

ElGamal Message Exchange

B encrypts a message to send to A computing

- Represent message M in range: $0 \leq M \leq q - 1$
  - Longer messages must be sent as blocks
- Choose a random integer k with $1 \leq k \leq q - 1$
- Compute a one-time key $K = {Y_A}^K \bmod q$
- Encrypt and send M as a pair of integers (C_1, C_2) where
  - $C_1 = a^k \bmod q$; $C_2 = KM \bmod q$

A then recovers message by:

- Recovering key K as $K = {C_1}^{X_A} \bmod q$
- Computing M as $M = C_2K^{-1} \bmod q$
- Proof: (K: same as in Diffie-Hellman; K^{-1} multiplicative inverse in GF(q))

ElGamal Example

Use field GF(19) q = 19 and a = 10 (a primitive root)

Alice computers her key:

- A chooses $X_A = 5$ and computes $Y_A = 10^5 \bmod 19 = 3$

Bob sends message M=17 as (11, 5) by:

- Choosing random k = 6
- Computing $k = {Y_A}^k \bmod q = 3^6 \bmod 19 = 7$
- Computing $C_1 = a^k \bmod q = 10^6 \bmod 19 = 11$
- Computing $C_2 = KM \bmod q = 7*17 \bmod 19 = 5$

Alice recovers original message by computing:

- Recover $k = {C_1}^{X_A} \bmod q = 11^5 \bmod 19 = 7$

ElGamal Long Message Exchange

Longer Messages must be sent as blocks, and a unique value of k should be used for each block

- Otherwise, once one plaintext block, ex M_1, is known by attackers, other can be computed

$$C_{1,1} = \alpha^k \bmod q;\ C_{2,1} = KM_1 \bmod q$$

$$C_{1,2} = \alpha^k \bmod q;\ C_{2,2} = KM_2 \bmod q$$

Then,

$$\frac{C_{2,1}}{C_{2,2}} = \frac{KM_1 \bmod q}{KM_2 \bmod q} = \frac{M_1 \bmod q}{M_2 \bmod q}$$

If M_1 is known, then M_2 is easily computed as

$$M_2 = (C_{2,1})^{-1}\, C_{2,2}\, M_1 \bmod q$$

Figure 2: Example of Long Message being able to be attacked

Elliptic Curve Cryptography (ECC)

The key length of RSA has increased over the years

- A heavier processing load, especially on small devicess

ECC offers same security as RSA with a smaller key size

ECC is newer, but not as well analyzed

Can provide key exchange, encryption, digitical signature

Involves the use of an elliptic curve equation defined over a finite field (variables and coefficients are finite)

Comparable Key Size in terms of Computation Effort for Cryptanalysis

Symmetric key algorithms	Diffie-Hellman, Digital Signature Algorithm	RSA (size of *n* in bits)	ECC (modulus size in bits)
80	L = 1024 N = 160	1024	160–223
112	L = 2048 N = 224	2048	224–255
128	L = 3072 N = 256	3072	256–383
192	L = 7680 N = 384	7680	384–511
256	L = 15,360 N = 512	15,360	512+

Figure 3: Diagram of Algorithm strength

PRNG Algorithms Based on Asymmetric Ciphers

Asymmetric Encryption Algorithms produce apparently random output.

- Hence can be used to build a pseudorandom number generator (PRNG)
- Much slower than symmetric algorithms
- Hence only used to generate a short pseudorandom bit sequence

PRNG Based on RSA

Micali-Schnorr PRNG using RSA:

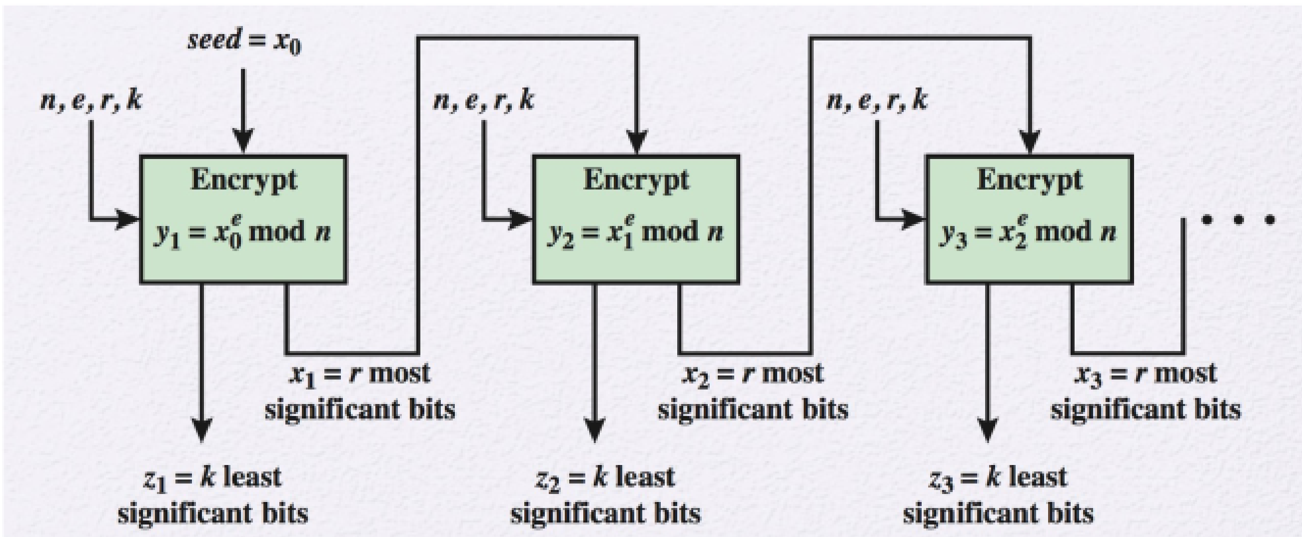

Figure 4: Diagram of RSA with PRNG

Similar to OFB mode used as PRNG

A portion of output is used as feedback, ensuring unpredictability

Six Requirements in Micali-Schnorr PRNG

1. n = pq (This is from RSA)
2. $1 < e < \phi(n)$, GCD(e, ϕ(n)) = 1 (This is from RSA)
3. re $\geq 2 \cdot N$, N = floor($\log_2$n) + 1
   - Ensures exponentiation requires a full modular reduction
4. r $\geq 2 \cdot$ strength
   - Strength is a value in {112,128,192,256}
   - 2^{strength} is the amount of work to break the security
5. k, r are multiple of 8 (Ease of implementation)
6. k $\geq$ 8; r + k = N (All bits are used)
   - r should be large to meet the requirements of 3 and 4
   - k should be large to get mroe bits in each encryption