Notes by Marat Boshernitsan
©1998 Marat Boshernitsan and Guy Blelloch

- Some group theory
- RSA summary
- ElGamal
- Probabilistic Encryption (Blum-Goldwasser)
- Quantum Cryptography
- Kerberos case Study

1 Some group theory

1.1 Groups: Basics

A group is a set G with binary operation $*$ defined on G, such that the following properties are satisfied:

1. **Closure.** For all $a, b \in G$, we have $a * b \in G$.
2. **Associativity.** For all $a, b, c \in G$, we have $(a * b) * c = a * (b * c)$.
3. **Identity.** There is an element $I \in G$, such that for all $a \in G$, $a * I = I * a = a$.
4. **Inverse.** For every $a \in G$, there exists a unique element $b \in G$, such that $a * b = b * a = I$.

For example, integers $\mathbf{Z}$ under the operation of addition form a group: identity is 0 and the inverse of a is $-a$.

1.2 Integers modulo n

For prime p, Z_p is a group of integers modulo p:

- $Z_p \equiv \{1, 2, 3, ..., p-1\}$
- $* \equiv$ multiplication mod p
- $I \equiv 1$
- $a^{-1} \equiv a * a^{-1} = 1 \pmod{p}$

Example: $Z_7 = \{1,2,3,4,5,6\}$. Note that if p is **not** a prime, Z_p is not a group under $*$, because there isn't necessarily an inverse. All encryption algorithms assume existence of an inverse. So, for n not prime, we define:

- $Z_n^* \equiv \{m \ : \ 1 \le m < n, \ gcd(n,m) = 1\}$ (i.e. integers that are relatively prime to n)
- $* \equiv$ multiplication mod n
- $I \equiv 1$
- $a^{-1} \equiv a * a^{-1} = 1 \pmod{p}$

Example: $Z_{10}^* = \{1,3,7,9\}$, $1^{-1} = 1$, $3^{-1} = 7$, $7^{-1} = 3$, $9^{-1} = 9$. It is not too hard to show that the size of Z_n^* is $\prod_{p \in P} p\left(1 - \frac{1}{p}\right)$ where P are the primes dividing n, including n if it is prime. If n is a product of two primes p and r, then $|Z_n^*| = pr(1-1/p)(1-1/r) = (p-1)(r-1)$.

1.3 Generators (primitive elements)

For a group G, element $a \in G$ is a **generator of** G if $G = \{a^n \ : \ n \in Z\}$ (i.e. G is "generated" by the powers of a). For Z_7 we have:

x	x^2	x^3	x^4	x^5	x^6
1	1	1	1	1	1
2	4	1	2	4	1
3	**2**	**6**	**4**	**5**	**1**
4	2	1	4	2	1
5	**4**	**6**	**2**	**3**	**1**
6	1	6	1	6	1

Here, 3 and 5 are the generators of Z_7 while 1, 2, 4 and 6 are not since none of them will generate 3. For Z_{10}^* we have:

x	x^2	x^3	x^4
1	1	1	1
3	**9**	**7**	**1**
7	**9**	**3**	**1**
9	1	9	1

So, 3 and 7 are the generators of Z_{10}^*. Note that any element to the power of the size of the group gives identity. In fact, there's a theorem stating that this is always true:

Theorem 1 *For all finite groups G and for any $a \in G$, $a^{|G|} = I$, where I is the identity element in G.*

From this theorem, it follows that for $a \in Z_p$, $a^{|Z_p|} = a^{p-1} = 1$, and for two primes p and q and $a \in Z_{pq}^*$, $a^{|Z_{pq}^*|} = a^{(p-1)(q-1)} = I$. The latter fact is essential for RSA.

1.4 Discrete logarithms

If g is a generator of Z_p (or Z_p^* if p is not prime), then for all y there is a unique x such that $y = g^x \pmod p$. We call this x the **discrete logarithm** of y, and use the notation $\log_g(y) = x \pmod p$.

For Example: in Z_7, $\log_3(6) = 3$, but $\log_4(5) = \emptyset$ (no inverse), and $\log_4(2) = \{2, 5\}$ (multiple inverses) because 4 is not a generator of Z_7.

2 Public Key Algorithms (continued)

2.1 RSA algorithm (continued)

Here are the 5 steps of the RSA algorithms, again

1. Pick two random large primes p and q (on the order of a few hundred digits). For maximum security, choose p and q to be of the same length.
2. Compute their product $n = pq$.
3. Randomly choose the encryption key e, such that e and $(p-1)(q-1)$ are relatively prime. Then, find the decryption key d, such that $ed \bmod (p-1)(q-1) = 1$. I.e., $ed = k(p-1)(q-1) + 1$, for some integer k. The numbers e and n are the public key, and the numbers d and n are the private key. The original primes p and q are discarded.
4. To encrypt a message m: $c = m^e \pmod n$.
5. To decrypt a cipher c: $m = c^d \pmod n$.

Why does this work? Note that $m = c^d = (m^e)^d = m^{ed} = m^{k(p-1)(q-1)+1} = (m^{(p-1)(q-1)})^k m$ (all operations mod$n$). But m is relatively prime to n (because $n = pq$ and p, q are primes), and so m is in Z_n^*. By the theorem in Section 1.3, m to the power of the size of Z_n^* is 1, so $(m^{(p-1)(q-1)})^k m = 1^k m = m$. In practice, we pick small e to make m^e cheaper.

The performance figures of RSA (relative to block ciphers) are fairly unimpressive. Running on Sparc 2 with 512-bit n, 8-bit e, the encryption throughput is 0.03 secs/block, which translates into 16 Kbits/sec. The decryption throughput is even worse with 0.16 secs/block (4 Kbits/sec). Such speeds are too slow even for a modem. With special-purpose hardware (circa 1993), we can boost the throughput to 64 Kbits/sec which is fast enough for a modem, but is slow for pretty much everything else. In comparison, IDEA block cipher implemented in software can achieve 800 Kbits/sec. Because of its slowness, RSA is typically used to exchange secret keys for block ciphers that are then used to encode the message.

In the "real world", the RSA algorithm is used in X.509 (ITU standard), X9.44 (ANSI proposed standard), PEM, PGP, Entrust, and many other software packages.

2.2 Factoring

The security of RSA is based on the difficulty of factoring large numbers. The state of the art in factoring is as follows:

- Quadratic sieve (QS): $T(n) = e^{(1+O(1))(\ln(n))^{\frac{1}{2}}(\ln(\ln(n)))^{\frac{1}{2}}}$. This algorithm was used in 1994 to factor 129-digit (428-bit) number. It took 1600 Machines 8 months to complete this task.
- Number field sieve (NFS): $T(n) = e^{(1.923+O(1))(\ln(n))^{\frac{1}{3}}(\ln(\ln(n)))^{\frac{2}{3}}}$. This algorithm is about 4 times faster than QS on the 129-digit number and is better for numbers greater than 110 digits. It was used in 1995 to factor 130 digit (431-bit) number with approximately the same amount of work as taken by QS on 129-digits.

2.3 ElGamal Algorithm

The ElGamal encryption scheme is based on the difficulty of calculating discrete logarithms. The following are the steps of ElGamal:

1. Pick a prime p.
2. Pick some g, such that g is a generator for Z_p.
3. Pick random x and calculate y, such that $y = g^x \pmod{p}$. Now, distribute p, g, and y as the public key. The private key will be x.
4. To encrypt, choose a random number k, such that k is relatively prime to $p-1$. Calculate $a = g^k \pmod{p}$ and $b = y^k m \pmod{p}$. The cypher is the pair (a, b).
5. To decrypt, calculate $m = \frac{b}{a^x} \pmod{p}$.

Why does ElGamal work? Because $m = \frac{b}{a^x} = \frac{y^k m}{g^{kx}} = \frac{g^{kx} m}{g^{kx}} = m \pmod{p}$. To break ElGamal, one would need to either figure out x from y or k from a. Both require calculating discrete logs which is difficult.

Performance-wise, ElGamal takes about twice the time of RSA since it calculates two powers. It is used by TRW (trying to avoid the RSA patent).

3 Probabilistic Encryption

One of the problems with RSA encryption is that there is a possibility of gaining some information from public key. For instance, one could use public key E_{public} to encrypt randomly generated messages to see if it is possible to get some information about the original message.

This problem can be remedied by mapping every message m to multiple c randomly. Now, even if cryptoanalist has c, m, and E_{public}, she cannot tell whether $c = E_k(m)$ (i.e. k was used to encrypt m).

One such probabilistic encryption scheme, Blum-Goldwasser, is based on generating strings of random bits.

3.1 Generation Random Bits

This method is called Blum-Blum-Shub (BBS) and is based on the difficulty of factoring: given a single bit, predicting the next bit is as hard as factoring. Here are the steps:

1. Pick primes p and q, such that $p \bmod 4 = q \bmod 4 = 3$.
2. Calculate $n = pq$ (called Blum integer).
3. For each m we want to encrypt, choose random seed x that is relatively prime to n ($gcd(x, n) = 1$).
4. The state of the algorithm is calculated as follows:
   - $x_0 = x^2 \pmod n$
   - $x_i = x_{i-1}^2 \pmod n$
5. The i^{th} bit is then calculated by taking the least-significant bit of x_i.

Note that now we have $x_i = x_0^{2^i \bmod (p-1)(q-1)} \pmod n$ and $x_0 = x_i^{-2^i \bmod (p-1)(q-1)} \pmod n$. In other words, given x_0 and th prime factors p and q of n it is not hard to generate any i^{th} bit and vice versa. We shall now see how these properties can be used to construct an encryption scheme.

3.2 Blum-Goldwasser

Blum-Goldwasser is a public key probabilistic encryption scheme using BBS. The public key is n, the private key is the factorization of n, i.e. p and q. The following are the steps to encrypting a message:

1. For each bit m_i of the message ($0 \leq i < l$) compute $c_i = b_i \otimes m_i$ where b_i is i^{th} bit of BBS (as in usual stream cipher).
2. Append x_l to the end of message, where l is the length of ciphertext in bits.

To decrypt, we use the private key p and q to generate x_0 from x_l ($x_0 = x_l^{-2^l \bmod (p-1)(q-1)}$) and then use the BBS formula to regenerate x_i, and hence b_i. The security of this scheme is based on difficulty of calculating x_0 from x_l without knowing the factorization of n.

Note that this encryption scheme achieves the desired probabilistic property: each time we code the same message with the same key we get different cipher (provided x is chosen randomly for each message). Furthermore the extra information we need to send (x_l) is minimal.

4 Quantum Cryptography

[This section has been expanded based on the material from
`http://eve.physics.ox.ac.uk/NewWeb/Research/crypto.html`]

Quantum cryptography makes use of the fact that measuring quantum properties can change the state of a system, e.g. measuring the momentum of a particle spreads out its position. Such properties are called *non-commuting observables*. This implies that someone eavesdropping on a secret communication will destroy the message making it possible to devise a secure communication protocol.

Photon polarization is one such property that can be used in practice. A photon may be polarized in one of the four polarizations: 0, 45, 90, or 135 degrees. According to the laws of quantum mechanics, in any given measurement it is possible to distinguish only between rectilinear polarizations (0 and 90), or diagonal polarizations (45 and 135), but not both. Furthermore, if the photon is in one of the rectilinear polarizations and someone tries to measure its diagonal polarization the process of measurement will destroy the original state. After the measurement it will have equal likelihood of being polarized at 0 or 90 independent of its original polarization.

These properties lead to the following is a key exchange protocol based on two non-commuting observables (here, rectilinear and diagonal polarizations):

1. Alice sends Bob photon stream randomly polarized in one of 4 directions.

2. Bob measures photon polarization, choosing the kind of measurement (diagonal or rectilinear) at random. Keeping the results of the measurement private, Bob then tells Alice (via an open channel), the kinds of measurement that he used.

3. Alice tells Bob which of his measurements are correct (also via an open channel).

4. Bob and Alice keep only the correct values, translating them into 0's and 1's.

Although this scheme does not prevent eavesdropping, Alice and Bob will not be fooled, because any effort to tap the channel will be detected (Alice and Bob will end up with different keys) and the eavesdropper will not have the key either (since she is very unlikely to have made the same measurements as Bob).

While practical uses of quantum cryptography are yet to be seen, a working quantum key distribution system has been built in a laboratory at the IBM T. J. Watson research center. This prototype implementation can transmit over admittedly modest length of 30cm at rate it 10 bits/second. Nevertheless, as new photon sources, new photo-detectors and better optical fibers are being built, we shall see quantum cryptography becoming more practical.

5 Kerberos (A case study)

Kerberos is a trusted authentication protocol. A Kerberos service, sitting on the network, acts as a trusted arbitrator. Kerberos provides secure network authentication, allowing

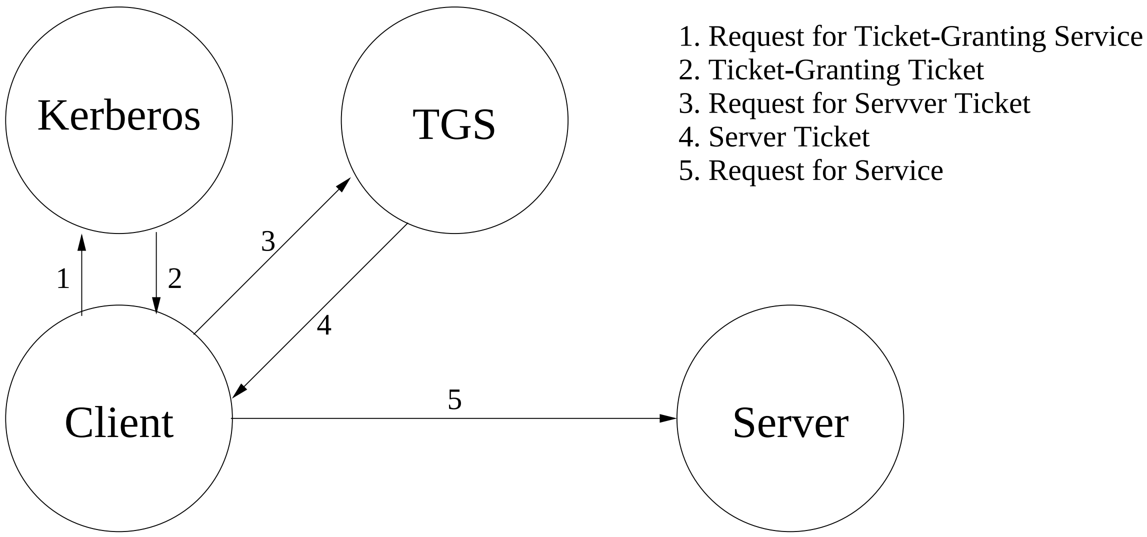

Figure 35: Kerberos authentication steps.

c	client
s	server
a	client's network address
v	beginning and ending validity time for ticket
t	timestamp
K_x	x's secret key
$K_{x,y}$	session key for x and y
$\{m\}K_x$	m encrypted in x's secret key
$T_{x,y}$	x's ticket to use y
$A_{x,y}$	authenticator from x to y

Table 8: Kerberos Notation

a person to access different machines on the network. Kerberos is based on symmetric cryptography (DES).

5.1 How Kerberos Works

The basic Kerberos authentication steps are outlined in Figure 35. The client, a user or an independent software program, requests a *Ticket-Granting Ticket (TGT)* from Kerberos server located on a trusted and secure machine. Kerberos replies with this ticket, encrypting it with client's secret key. To use the requested service, the client requests a ticket for that server from *Ticket-Granting Sever (TGS)*. TGS replies to client with the server ticket, which the client uses along with authenticator to request services from the server.

5.2 Tickets and Authenticators

Kerberos uses two kinds of credentials: tickets and authenticators. The tickets have the following format (refer to Table 8 for notation):

$$T_{c,s} = s, \{c, a, v, K_{c,s}\}K_s \tag{1}$$

A ticket is good for a single server and a single client. It contains the client's name and network address c, a, the server's name s, a timestamp v, and a session key $K_{c,s}$. Once the client gets this ticket, he can use it multiple times to access the server – until the ticket expires.

The authenticator takes this form:

$$A_{c,s} = \{c, t, key\}K_{c,s} \tag{2}$$

The client generates an authenticator each time he wants to use service on a server. The authenticator contains the clients name c, a timestamp t, and an optional additional session key. An authenticator can only be used once, but the client can regenerate authenticators as needed.

5.3 Kerberos messages

There are five Kerberos messages:

1. Client to Kerberos: c, tgs. This message is sent whenever the client wants to talk to Ticket-Granting Server. Upon receipt of this message, Kerberos looks up client in its database and generates the session key to be used between the client and TGS.

2. Kerberos to client: $\{K_{c,tgs}\}K_c, \{T_{c,tgs}\}K_{tgs}$. There are two parts in this message, one containing session key to be used between client and TGS ($K_{c,tgs}$) encrypted with clients key (K_c), and the other containing TGT ($T_{c,tgs}$) encrypted with TGS's secret key (K_{tgs}). The client cannot decrypt this latter part, but uses it to talk to TGS. Note that Kerberos does not communicate with TGS directly and the tickets have limited lifetime.

3. Client to TGS: $\{A_{c,s}\}K_{c,tgs}\{T_{c,tgs}\}K_{tgs}$. This is the message sent by the client to TGS. The first part of this message contains information that proves client's identity to TGS. The second part is precisely the TGT Kerberos sent to client.

4. TGS to client: $\{K_{c,s}\}K_{c,tgs}\{T_{c,s}\}K_s$. This message is similar to Kerberos' response to clients initial request. It contains the session key to be used between client and the server as well as the ticket for the client to use when requesting service. These also have limited lifetimes, depending on the kind of service requested.

5. Client to server: $\{A_{c,s}\}K_{c,s}\{T_{c,s}\}K_s$. With this message the client authenticates itself to the server as well as presents it with the ticket it received from TGS.