6.915 Computer and Network Security **Fall Term, 1995**

Lecture 7 : September 28, 1995

Lecturer: Ron Rivest *Scribe: Lucy Phan*

Correction on Elgamal: Given a prime p and g an element modulo p. In order to check if g is a generator in polynomial time, you **do** need to know the factorization of $p-1$. In fact generators are the numbers g such that

$$g^q \neq 1$$

for all q prime divisors of $p-1$.

Topics covered today: Key agreement, Key Escrow, Oblivious Transer, Zero-Knowledge Proofs of Knowledge

1 Key Agreement

In this section we are going to talk about ways in which two parties can agree on a key to use for encryption.

1.1 Basic Diffie-Hellman

Let p be a large prime publicly known and g a generator modulo p.

1. Alice picks a random number mod $p-1$, which we will call x. This number is kept secret from the adversary. She then sends Bob g^x.

$$A \xrightarrow{g^x} B$$

2. Bob picks a random number mod $p-1$, which we will call y. This number is kept secret form the adversary. He then sends Alice g^y.

$$A \xleftarrow{g^y} B$$

3. Now both Alice and Bob can use g^{xy} as their session key. Note that Alice gets g^y from Bob. She already knows x. The same applies for Bob.

Notice that if Alice and Bob's public keys are g^x and g^y, respectively, the process is non-interactive

There are problems with the Basic Diffie-Hellman:

- Only works if the adversary is passive.
- There is forward secrecy but **no** authentication.

This makes Diffie-Hellman vulnerable to "man in the middle" attack:

1. Eve, for example, can pretend that she is Bob to get g^x from Alice. She can then send Alice some $g^{y'}$ which Alice would think is from Bob.

2. Eve then pretends that she is Alice and gets g^y from Bob. She can then send Bob some $g^{x'}$ which Bob would think is from Alice.

3. In this way Eve can intercept messages from Alice and Bob without them knowing it.

1.2 Station To Station (STS)

This protocol comes from Diffie/Oorschot/Wiener.

Assume each party has a public key used for signing only. Also assume the existence of a global prime p and generator g.

1. Alice sends Bob g^x. (Note x is a random number mod $p-1$ chosen by Alice.)

$$A \xrightarrow{g^x} B$$

2. Bob sends Alice back g^y along with $E(K, sig_B(g^x, g^y))$ (Note y is a random number mod $p-1$ chosen by Bob. K is the session key, g^{xy}.)

$$A \xleftarrow{g^y, E(K, sig_B(g^y, g^x))} B$$

3. Alice is now assured that Bob has g^x and that the g^y she received is from Bob since it contains Bob's signature. Alice then sends Bob a confirmation.

$$A \xrightarrow{E(K, sig_A(g^x, g^y))} B$$

4. Bob now knows that Alice received g^y and that the g^x that he received is from Alice.

STS eliminates the middle man while retaining perfect forward secrecy.

Key Idea: Only use your long term secret key for signing.

2 Key Escrow

There is an increasing need to protect various information infrastructure using encryption. However the government seeing the potential of misuse of this technology, is pushing key escrow so that it can gain access to private information. This supposedly voluntary act does not make sense if the aim is to deter criminals since they will find ways around it.

In any case, the current proposal for Clipper is as follows:

- It uses a secret algorithm (Skipjack) with an 80-bit key.
- It is hardware based. The crypto chips will be produced by the government.
- Each chip has a unique 32-bit serial number (UID), an 80-bit Family Key (KF), and an 80-bit Unit Key (KU).
- When two chips want to communicate, a session key (KS) is arranged with external techniques. Included in the ciphertext stream is a Law Enforcement Access Field (LEAF).
- When the chips are manufactured, their unit keys are split and given to two agents. The $\oplus$ of these two pieces will produce the complete unit key.
- Any agency wanting the KU will have to obtain it from both escrow agents. The KU is then used to initialize a decrypt processor, which can obtain KS since it knows both KF and KU.

How the LEAF is computed:

- The first 80 bits is the session key (KS) encrypted with the unit key (KU).
- The next 32 bits is the unique serial number (UID).

- The last 16 bits are 0's (used to provide redundancy).
- the above 128 bits are then encrypted using the family key (KF).

Other proposals:

1. Secret sharing of your private key.
2. Limit the time law enforcement agencies are able to access the information.
3. Make law enforcement agency work to decrypt the message by on giving them part of the key
4. Have translucent cryptography wherein law enforcement agency can only see part of the traffic.

3 Oblivious Transfer

This is a protocol where one party (Alice) can transfer a message to another party (Bob) in such a way that

- Bob has the probability of 1/2 of actually receiving the message.
- Alice does not know if Bob received the message but Bob does.

A possible solution for this protocol is as follows:

1. Let p be a large prime, and g be a generator mod p. (Finding the discrete logs mod p should be hard.)
2. Assume that it is infeasible for anyone to find the discrete log of 2.
3. Bob publishes two numbers: v and $v' = 2v$. Notice that because of the previous point he knows the discrete log of exactly one of them.
4. To send a message to Bob via oblivious transfer, Alice picks v or v', and encrypts the message with it for example using ElGamal. Say that Alice picked v, then

$$(c_1, c_2) = (g^k, m + v^k)$$

5. If Alice picks the correct one (which Bob knows the log of) then Bob can read the message. Otherwise he can't.

This method can be generalized to other probabilities of success. (Note: Anything of the form $a/2^b$ should work.)

This method is applicable in translucent cryptography where each message is prefixed by a non-interactive oblivious transfer of the message key to the government with a probability of $\frac{1}{5}$. Then the government is able to read 20% of the messages. The appropriate percentage could be picked for example by Congress.

It can also be used in simultaneous exchange of

- secrets
- contract
- signatures
- mail/receipt

4 Zero-Knowledge Proofs of Knowledge

Suppose Alice wants to prove herself to Bob. That is she wants to identify that she is really Alice. One way of doing this is to prove to Bob that she knows something only she should know. For example a password. However in this case she is betraying her secret to Bob himself and to anyone who happens to listen on the conversation. Another approach would be to prove to Bob that she knows the seret key corresponding to a public key associated to her name. For example this could be done by Alice signing a random challenge sent by Bob. However how is Alice guaranteed that she does not leak any knowledge about her secret key? For example a malicious Bob could pick the challenge not at random but in some malicious way that causes some leaking of knowledge.

In this section we will present ways of proving knowledge of some information without revealing absolutely anything about the information itself. This is the reason why such proofs are called zero-knowledge proofs of knowledge.

For example let (n, e) be a RSA public–key. Assume that Alice wants to prove that she knows some m where $c = m^e \bmod n$ is her public key then she could do the following:

1. Alice sends Bob $y = r^e$ where r is a random number mod n.
2. Bob sends Alice either $b = 0$ or $b = 1$
3. Alice sends Bob $z = r * m^b$ that is either r or rm.
4. Bob then checks the result, that is if he sent $b = 0$ he checks that $z^e = y$, if he sent $b = 1$ he checks that $z^e = yc$.

Ths basic protocol being repeated many times (say 20.)

Why does this protocol work? In particular what does it achieve?

We have to prove the follwoing things:

1. The correct Alice who knows m should always pass the protocol. This property is usually called **completeness.** This is clearly true in our case.
2. If a bad Eve tries to impersonate Alice without knowing m she should fail with high probability. This property is usually called **soundness.** Notice that since Alice is prepared to answer both questions $b = 0$ or $b = 1$) then she must know m. For Eve instead the only possibility would be to guess what Bob is going to ask. She chooses r at random. If she thinks Bob is going to ask $b = 0$ then she behaves like Alice in the first message sending $y = r^e$. If she thinks Bob is going to ask $b = 1$ then she sends $y = r^e/c$. Her answer in the first message will always be the only e–root she knows, i.e. r. However if the protocol is repeated 20 times and Bob's choices are always randomly and independently chosen Eve will havea probability of convincing him equal to 2^{-20}. Notice thta by increasing the number of repetitions one can bring this probability as low as desired.
3. Finally we want that Bob learns nothing from his interaction with Alice. This is why this kind of protocols is called **zero–knowledge**. We have to argue that the transcript of the conversation with Alice reveals nothing to Bob. A way to argue this is to prove that Bob can construct such transcript on his own without interacting with Alice. Indeed Bob could (behaving like Eve) generate a transcript with the same identical probability distribution of the true conversations.