# CS 261: Crypto Protocols Cont., Elliptic Curve Crypto, ATMs

Scribe: David Kantola

November 21, 2011

## 1 Cryptographic Protocols Continued

### 1.1 TMN Protocol

This is a real protocol proposed in the literature for a special purpose. Two parties, Alice and Bob, want to use the help of $S$, a trusted third party, to establish a shared key. The special circumstances are that Alice and Bob are smartcards or other low end devices without much computing power, but $S$ is a server with significantly more computing power. While we usually think of public-key operations as slow, this protocol exploits the special case that encryption in RSA is extremely fast. So the two parties with little computing power use RSA encryption, while the server performs the slow decryption operation, then provides the parties with enough information to recover a shared key. The two parties first choose random $N_a$ and $k_{ab}$. Assume the parties already have the server's public key $k_s$, and they want to establish $k_{ab}$ as the shared key.

1. $A \rightarrow S : \{N_a\}_{k_s}$
2. $B \rightarrow S : \{k_{ab}\}_{k_s}$
3. $S \rightarrow A : N_a \oplus k_{ab}$

The server essentially uses $N_a$ as a one-time pad on $k_{ab}$, which allows Alice to recover $k_{ab}$ with a fast XOR operation, since $N_a \oplus (N_a \oplus k_{ab}) = k_{ab}$.

The first flaw in the protocol is that nothing authenticates Alice's or Bob's messages to $S$, so an attacker can spoof the first two messages. The second flaw can be exploited by two colluding malicious clients, $M$ and $M'$, who first learn $\{k_{ab}\}_{k_s}$ by eavesdropping on message 2 between Alice and Bob. Now that they have $k_{ab}$ encrypted under the server's public key, they would like to use the server as an oracle to decrypt the shared key for them. To do this, instead of generating a new nonce, $M$ can replay $\{k_{ab}\}_{k_s}$ to the server:

1. $M \rightarrow S : \{k_{ab}\}_{k_s}$
2. $M' \rightarrow S : \{N_{m'}\}_{k_s}$
3. $S \rightarrow M : k_{ab} \oplus N_{m'}$

Now $M$ and $M'$ can together recover $k_{ab}$. This attack was tricky to find, because it only appears in an obscure threat model that includes at least five parties, where two malicious parties are colluding.

To prevent the replay attack, the first two messages could be bound to the two parties by including their identities inside the message. Then the messages could be authenticated with signatures:

1. $A \to S: \{\{A, B, N_a\}_{k_s}\}_{k_{a^{-1}}}$

2. $B \to S: \{\{B, A, k_{ab}\}_{k_s}\}_{k_{b^{-1}}}$

3. $S \to A: N_a \oplus k_{ab}$

Now the attack fails because $M'$ can't sign $\{A, B, N_{m'}\}_{k_s}$ with Alice's private key.

### 1.2 Smash Protocol

In this published protocol, which is very similar to Needham-Schroeder, Alice and Bob know each other's public keys, and Alice wants to send a message to Bob. The first message is a challenge to Bob, proving that Bob knows his private key.

1. $A \to B: \{A, N_a\}_{k_b}$

2. $B \to A: \{N_a\}_{k_a}$

3. $A \to B: \{N_a, message\}_{k_b}$

The flaw is that nothing prevents someone from claiming to be Alice in the first message. Anyone, say Zorro, could have placed her name in the first message and encrypted it under Bob's public key:

1. $Z \to B: \{A, N_z\}_{k_b}$

2. $B \to A: \{N_z\}_{k_a}$

3. $Z \to B: \{N_z, message\}_{k_b}$

In Needham-Schroeder, Bob also supplied his own nonce in the second message, and then challenged Alice to decrypt it with her private key. This protocol left out the step of authenticating Alice.

### 1.3 Conclusion

These are some examples of published cryptographic protocols with flaws that went unnoticed for years. The point is that it can be very tricky to spot flaws in cryptographic protocols, and these are simple examples. The techniques used for analyzing cryptographic protocols are also useful for analyzing sophisticated protocols on the Web.

Cryptographic protocols for basic functionality such as key exchange or setting up a secure channel are basically solved problems. New cryptographic protocols still need to be created for new sets of requirements, for example, a customer making a payment to a store through a third-party payment processor.

## 2 Elliptic Curve Cryptography

Consider Diffie-Hellman key exchange, where Alice chooses a random value $a$ and Bob chooses a random value $b$:

1. $A \to B: g^a \bmod p$

2. $B \to A: g^b \bmod p$

Now both Alice and Bob compute the key $g^{ab} \bmod p$, where $p$ is some large prime.

If you recall group theory, it's not important that we're computing modulo a prime, it's important that we have a group. To have a group, we need a way to multiply two elements of a set that results in an element of the same set. For example, if we have two integers in the set of integers modulo $p$, we can multiply them, reduce modulo $p$, and that gives us another integer in the set of integers modulo $p$. Elliptic curve cryptography gives us another way of doing multiplication on a finite set. It has all the properties you might expect for multiplication. A finite set together with a multiplication operation that satisfies all these familiar properties is a group.

Now we can do Diffie-Hellman key exchange in another group by interpreting exponentiation as repeated multiplication. Thus, multiplication gives us an exponentiation operation. However, this won't be secure unless the mapping $a \mapsto g^a$ is a one-way mapping. This means that although it's cheap to compute $g^a$ from $a$, it's very difficult to recover $a$ given $g^a$. We believe that if $p$ is a large enough prime, $a \mapsto g^a \bmod p$ is a one-way mapping. Elliptic curve cryptography is another way of defining a multiplication operation where it's believed that if we have a large enough finite set, reversing the mapping is hard.

Why bother with elliptic curve cryptography instead of mod $p$ cryptography? The only reason is because elliptic curve cryptography is a little bit faster and saves some space. Multiplication mod $p$ is more expensive because we only believe it to be a one-way mapping when the prime is very large, say 2,000 bits. But if the prime is of intermediate size, say 300 bits, then there are sub-exponential time algorithms to reverse the mapping. So we need to make the prime large enough to beat these sub-exponential time algorithms as well as brute-force search.

No sub-exponential time algorithms are known to reverse the elliptic curve cryptography mapping. The fastest method anyone knows to break it is a brute-force approach. So now you don't need a prime of thousands of bits, you only need a finite set of effectively a few hundred bits. This means the ciphertexts are shorter and the multiplication operation is faster. Elliptic curve cryptography basically gives us a plugin replacement for multiplication.

Because the group of integers modulo $p$ has an addition operation, they have additional structure over the group defined by an elliptic curve. The sub-exponential time algorithms for reversing multiplication modulo $p$ use this additional structure, which isn't present in elliptic curves. For this reason, nobody has been able to find a sub-exponential time algorithm that beats brute-force search.

## 3 ATMs

ATMs require both a card with a magstripe and a PIN to provide access to an account. This two-factor authentication protects against lost cards and guessable PINs, but not both simultaneously.

### 3.1 Early Designs

ATMs used to need to be able to validate your identity without a network connection. Today this requirement is less relevant, but in the 1980s there were ATMs in convenience stores, for example, without a phone line connection. Since ATMs had to support offline operation, they needed to support offline identity validation.

#### 3.1.1 First Design

In this design, the card would store the account number and the PIN encrypted with DES using a symmetric key specific to the bank and stored in every offline ATM. In offline ATMs, the key would be protected with an enclosure designed to prevent tampering. By constrast, online ATMs would store the key in a mainframe offsite.

One attack on this scheme is to learn someone else's account number, then use a magstripe writer to replace your account number with theirs on your card. Since you know your own PIN, you can withdraw money from their account.

To use this attack, you need to learn other people's account numbers. Account numbers used to be printed on receipts, so they could be found in trash cans. A skimmer can also be attached to an ATM to read the magstripe, grabbing the account number. The account number is printed on checks too. Since account numbers aren't kept very secret, any scheme such as this that relies on their secrecy for security is broken.

A countermeasure to this attack is to bind the account number to the PIN by encrypting them together. But DES uses 64-bit blocks, which are too small to fit both the account number and the PIN, and banks didn't want to get into modes of operation. There were also problems with backward-compatibility.

Another attack is to gather enough encrypted PINs from cards to create a mapping from DES-encrypted PINs to actual PINs.

### 3.1.2 Second Design

In this second design, only the account number is stored on the card. The PIN is derived from the account number by first encrypting the account number with DES, then decimalizing the upper four resulting hex digits. The decimalization function maps digits 0-9 to themselves and a-f to 0-5.

This scheme is no longer used because the fact that users couldn't choose their own PINs provided a bad user experience. It also has a security problem because the decimalization function introduces a bias in PIN-generation, since the digits 0-5 are twice as likely as 6-9.

An attack on this scheme is to learn both the account number and PIN, then use this information to create a copy of the card. This breaks two-factor authentication, because the attacker never needs possession of the card to gain access to the account.

## 3.2 Current Design

In the current design, only the account number and an offset are stored on the card, and the DES key is no longer stored in ATMs. The PIN is derived from the decimalized, encrypted account number by adding the offset. Since an attacker needs to read the card's magstripe to obtain the offset, this scheme makes it more difficult to create a copy of the card. However, it inherits the weaknesses of DES as well as the bias in decimalization.

### 3.2.1 Attacks

In some early attacks, attackers built a fake ATM and located it in a public place. Since the fake ATM could read the magstripe and PIN, it provided enough information to create a copy of the card.

A more modern attack is to add a magstripe skimmer to an existing ATM that captures and stores the magstripe data of all cards inserted into the ATM. A cell phone may be placed near the skimmer to relay the data back to the attackers. To capture the PIN, attackers place a tiny video camera on the ATM pointed at the keypad. Kits are readily available to perform this kind of attack on ATMs. More sophisticated equipment is also available that encrypts the captured data so the risky job of placing the skimmers can be delegated without allowing workers to get access to the compromised accounts. This demonstrates growing conspiracy, collusion, and specialization in these kinds of attacks.

The skimmer attack is very hard to defend against because it destroys the trusted path between the cardholder and the ATM, exposing a fundamental flaw in the current scheme. It could be described as a protocol where the cardholder is interacting with the ATM, and the protocol is not resilient against interacting with a malicious ATM, but it's very hard to determine whether the ATM is malicious.

## 3.3 European System

The UK and Europe are beginning to standardize on a new system called Chip and PIN where the card no longer has a passive magstripe with data, but instead has a smartcard with contacts for power and data that stores a private key and can do cryptography. This provides a defense against a malicious ATM, since the smartcard can perform a cryptographic protocol with the ATM that provides mutual authentication. Because the card never reveals its private key, a malicious ATM isn't able to collect enough information to create a copy of the card.

There have been side-channel attacks that observe the card's power consumption to infer the sequence of instructions running on the card's CPU, which can reveal the private key. Current smartcards include defenses against these kinds of attacks.

The system does have security weaknesses because the card has no user interface, so malicious card readers can lie about the transaction amount to the user, sending a larger transaction amount to the card. Malicious card readers can also capture the PIN.