Dr. Ivan Pustogarov
Concordia University

Lecture 2 Notes

Cryptographic Protocols by Example

Slides 1-4. In this lecture, we will try to cover general principle when designing a cryptographic protocol. We will start with a few definitions, we will then try to build a simple protocol to exchange a shared key between two parties.

A protocol can be defined a set of rules or conventions that govern the exchange of information between two or more parties such as computers or humans. We are going to speak about computer protocols, but the term protocol is also applicable to scenarios where computers are not involved, for example, you probably heard of so called diplomatic protocol which defines steps and rules on how a diplomatic activity should be performed. Protocols executed by computer programs have very strict set of rules of course.

Cryptographic protocols are a subclass of protocols that use cryptographic techniques to achieve some security objective. For example, the goal might be to check the authenticity of a web-site. Other goal might be to establish a new session key for encryption, or you might want a protocol that allows you stay anonymous online.

You might remember during the previous lecture, when we spoke about security context, one of the components was a security goal. And there are a number of security protocols each trying to achieve a specific goal. You might remember Diffie-Hellman protocol which can be used to address key exchange. For anonymity, there is Tor whose protocol is described in Tor specification. Then you also might think about Bitcoin protocol which allows you to send and receive digital money. So again, for cryptographic protocols there is some security objective.

Slide 5. There are two logical components in a protocol. A protocol usually governs communication between two or multiple parties, so the first component is communication component which defines how messages are transferred from one principal to another. For example, a protocol might define at what stage what message is sent, whether to encrypt it, whether to add authentication tag and so on.

The second component is computational step. During a protocol run, each party maintains some kind of internal state. For example, if it is a key exchange protocol that involves several messages exchanges, then after each message exchange the internal state is updated using newly received information. This new information can also used in a some sort of computation to generate new values and send them to other parties. For example, in a challenge-response protocol, party A sends a challenge (a string or a value) to party B. Party B then encrypts it (computation), and sends the result of the encryption to party A.

Slide 6. There are many examples of widely used cryptographic protocols. For example, TLS (and its predecessor SSL) is used under the hood by browsers when you connect to an HTTPS web-site. IPSec protocol is used when you set up a VPN. PGP can be used for email encryption and setting up web of trust. There are many protocols based on Blockchain, protocols for electronic voting, authentication protocols such as Kerberos.

One important note: protocol specifications and parameters are usually not a secret. One of the reasons is interoperability: anyone should be able to implement the protocol and it should work with existing systems. The second reason is you obviously should not rely on hiding protocol details to achieve security. If you remember you should never expect to achieve security through obscurity.

Slide 7. Let's see where we can map security protocols in the hierarchy of security components. At the bottom, we have mathematical foundation such as number theory, computational complexity, probability theory, and others.

Then we have cryptographic primitives that make use of concepts and theorems from the underlying theories. At this level, we have public key encryption algorithms, symmetric key encryption algorithms, hash functions, digital signature algorithms, and so on.

Each of these cryptographic primitives solves a very specific problem and often relies on availability of some information. For example, symmetric key encryption algorithms assume that the symmetric key was somehow shared between two parties. Or in the case of public key encryption, we need to be able to verify that the ownership of public keys.

Because of these initial assumptions and the fact that in real-world scenarios we need to achieve multiple simultaneous goals (for example we want data confidentiality and data integrity at the same time) we need to combine those primitives into one system. We also need to make sure that those primitives, when combined, do not degrade security of individual components. Moreover, we need to specify how data is exchange between two or multiple parties that try to achieve some security goal. This is what usually specified in cryptographic protocols.

And finally, at the upper level in our hierarchy, we have implementation. This is where a developer takes a protocol or a specification and implements it using some programming language. The implementation is then deployed within some system. Examples where cryptographic protocols are used are Web browsers, File sharing applications, voice and data communication programs, and many others.

Slide 8. Designing a new cryptographic protocol is not easy. First, when you design a protocol, the goals might turn to be more subtle that initially thought. For example, in electronic voting protocols, we can first think that the only thing we need is to guarantee that each voter is authenticated and votes only once. We then can design the protocol based on these two goals. But then it turns out that a) you also need to give voters the ability to check if their votes were counted properly; b) you also might want to prevent people from checking votes of other voters.

During the last lecture we talked a bit about threat model and attacker's capabilities. Capturing assumptions about the attacker is not trivial and requires a lot of experience. For example you might try to design a protocol for a bank, and it is assumed that this protocol will only be used in the bank's internal network, so you might want to assume that you can trust the employees. This of course not the case.

The protocol might also work under hostile environment. For example consider anonymity protocols such as Tor or I2P. We will discuss them in detail in the future lectures but in a very simplistic way these protocols work as follows. These systems have a large number of relay servers (or proxy servers) run by volunteers. And in order to hide someone's traffic these systems chain together several relays, usually three. Such systems work under the assumption that the attacker does not control or monitor the network infrastructure between the relays. The problem with this assumption is that the relay servers might reside inside one or several autonomous system that fall under the same jurisdiction. In this case the internet service providers might be compelled by the government to monitor and disclose the network traffic. Note: An autonomous system is a part of the Internet infrastructure controlled by the same provider.

Slide 9 When we think about attackers, we can think about them at different levels. First we need to have a clear understanding about what we are trying to protect from the attacker (or in other words what information the attacker will try to steal or corrupt.) This in turn will define attacker's methods and techniques that he/she will try to use. Not all methods will be available to all attackers: this depends on the attacker's capabilities, such computing resources, skills, knowledge, availability of specific software, how many people are going to try to break your protocol, is it solo hacker or it's a well-prepared criminal group. You also have different funding levels. For example a government-supported group will have more money they can spend on computing resources and personnel. More money also influences attackers' determination. Finally you need to understand if the attack can come from an insider. Such attackers will have an advantage as they might have access to user's credentials and physical access to the network.

Slide 10. Here are a few example of types of attacker it might be useful to think about. They all differ in motivation, available resources and funding levels. There are foreign intelligence agencies and government-funded groups which are the most powerful attackers with lots of capabilities. There are cyber-terrorists and politically motivated adversaries. There are groups who target companies' trade secrets. There are organized crime groups whose goal is stealing money. There are also individual hackers and malicious insiders (for example unhappy employees). And finally you also need to think about non-malicious employees who, due to the lack of security training, might inadvertently expose your systems to hackers.

Slide 11. The way the attacker will try to break into the system will vary too. The attacker might try to steal confidential information, for example intercept password during login. The attacker also might want to modify data in transit, for example if the transmitted data is unencrypted and not integrity protected. For example, in the case of a financial transaction, the attacker might be able to change the recipient's account. In other cases, the attacker might want to impersonate other participants (for example to send phishing emails.) Finally, the attacker's goal might be to simply block certain communication, for example to prevent a Bitcoin transaction.

Slides 12-13. Once we have an idea about attacker's goals we can think about corresponding security goals. If the attacker is interested in stealing private information, it is a threat to data confidentiality. If one of our security goals is data confidentiality, then we need add corresponding mechanisms, for example data encryption. If the attacker is trying to modify the data in transit, this is a threat to data integrity, and we need to add message integrity codes. Impersonation is a threat to data origin authentication, and if the attacker wants to block certain communication, it is a threat to service availability.

<u>Slide 14-15.</u> Let's see now try to use this knowledge to design a simple key establishment protocol. Our goal is exchange a session key K_{AB} between two users: Alice and Bob. We will assume that there is a trusted server S which will generate this shared key and distribute it to Alice and Bob. So overall in our protocol we have three parties Alice (A), Bob (B), and Server (S).

<u>Slide 16.</u> Before we start designing the protocol we need to set specific security goals. First, after the end of the protocol run, Alice and Bob should know the shared key K_{AB}. Second, we want to make sure that nobody else except for Alice, Bob, and the server knows the key. Third, since we are trying to exchange a sessions key, we have to be sure that this key is fresh (i.e. it's a newly generated key for the current session only.)

A side note about session keys. Usually communicating parties will have some sort of long-term keys or master keys. Using the same keys for every communication sessions has some important security implications. First, if the master key is compromised then the attacker will be able to decrypt all previous and future communications. Second, if you encrypt all the data with the same key, the attacker will have more data for a potential cryptanalytic attack. Using a new session key for each new session makes the consequences of loosing this key less dangerous.

<u>Slide 17.</u> Let's design the first version of our protocol. In this version, we can think about three steps. During the first step, Alice sends her and Bob's ID to the server. The server then generates a new key K_{AB}, and sends it back to Alice. Alice then sends this key along with her ID to Bob. Note that in this version of the protocol all communications are not encrypted. [The communication steps are shown at the right side of the slide.]

<u>Slide 18.</u> Let's now ask ourselves a few questions. Which of the security goals did we achieve with this protocol? Remember that G1 was A and B must know the shared key. G2 was no one else should be able to learn the key except for the Alice, Bob, and the Server. And third, Alice and Bob should be able to tell if the key is fresh.

Bob and Alice negotiated the key so we were able to achieve the first goal. But what about the second goal? Did we guarantee that no one else is able to learn the shared key? The correct answer is it depends. We might say that since all our messages are unencrypted, the attacker can intercept and read them. This is true if we are going to use public communication channel such as Internet so that an attacker can intercept our messages. But what if all the communications are going through a secure channel, for example there is a dedicated physical line between each of the parties? So in general whether we achieved the second goal depends on the attacker's capabilities and our resources.

Before we design an improved version of the protocol, let's introduce some notation that we are going to use. For public key encryption of some message M with user's A public key we use $E_A(M)$. For symmetric key encryption of message M with key K , we are going to use $\{M\}_K$. For MAC of a message M under key K, we will $MAC_K(M)$. And for digital signature generated by user A, we will use $Sig_A(M)$.

Slide 19. When we designed the first version of our protocol we missed one important step: threat model. Without knowing what an attacker can do, it's quite hard to add security measures. So we need to think what kind of attackers we want to protect the information from. And specifically, what powers the attacker has and what his limitations are. We also need to think how much resources we need to protect our system.

Slide 20. Let's now add some assumptions about the attacker. We are going to assume that we are using a public shared communication channels such as Internet and attackers have capabilities to intercept all protocol messages during any protocol run. With this threat model and the first version of the protocol, goal #2 is not achieved.: sending data over unencrypted channel leaks the key. So what capabilities do we need to achieve the confidentiality goal? Let's assume that Alice and Bob each has a long term encryption key with the server and let's redesign the protocol.

Slide 21. In the second version of the protocol, Alice sends a request to the Server, providing her and Bob's ID. The server generates a new encryption key K_{AB} and a) encrypts the key with Alice's long term key, and b) separately encrypts the key with Bob's long-term key. The server then sends this two encrypted messages back to Alice. Alice decrypts her message and learns the new session key. She also send the message encrypted with Bob's long term key to Bob along with her ID. Once Bob receives the message, he can decrypt it and learn the shared key.

What goal did we achieve now with the new version of the protocol? In this case (assuming we are using strong encryption and the long term keys were not compromised), we achieved data confidentiality in the sense that no one except for

Alice, Bob, and the server has access to the new session key. We extended our threat model and added the corresponding protection and our protocol looks better now.

Slide 22. Let's now extend our threat model. So far we considered a passive attacker who can only listen and record transmitted messages. Let's ``promote'' this attacker to active attacker. In this case, if he/she is in control of a part of the network infrastructure, he/she can also alter and re-route protocol messages. By altering we mean that he can insert new messages, remove some of the messages so that protocol participants don't receive them, or modify some of the messages during the transmission.

Within this updated threat model, let's think what kind of attacks an attacker can mount against our protocol. For example if attacker Carol, denoted as C, controls a portion of the network infrastructure she can intercept and modify packets destined to Bob during step three of the protocol. She can then substitute Alice's ID in the corresponding message with some other id D. In this way, while both Alice and Bob share a session key, Bob will think that the request came from a different user D, and not from Alice. This will effectively disrupt the protocol.

Slide 23. Let's look at yet another example of an attack that Carol can launch against the protocol. If C is in between Alice and Bob, then she can intercept and replace packets between them. This attack setting is called Man in the Middle Attack. Once she sees Alice initiating the protocol by sending her and Bob's ID to the server, Carol can intercept this message, and send another message to the server instead, containing Alice's and Carol's ID. The server will think that it needs to generate a new session key for Alice and Carol and reply to Carol with the session key encrypted with Alice's long-term key and Carol's long term key. Carol then will forward this message to Alice.

At this point, Alice thinks that she is talking to the server S and not to Carol, and will believe that the second portion of the message contains the session key encrypted with Bob's long term key. Alice will then send this part to Bob. At this step, Alice believes that she is exchanged a new session key with Bob while in reality she will be using the same key as Carol. And if Carol is able to further intercept packets, she can impersonate Bob.

Slide 25. Were we able to achieve our security goals with the new protocol version? Not under the updated threat model where an attacker can intercept and modify messages. In the last attack, the session key was exposed because even though Alice and Bob received a session key generated by server S, this session key was not meant for Bob. Let's try to fix it and redesign our protocol again!

Slide 26. In the next version, let's make the server encrypt not only the session keys but also users' identifiers. In this case, during the first step, Alice sends her and Bob ID to the server. The server generates a new session key, but before encrypting it with Alice's and Bob's long term keys, the server: a) appends Bob's ID to the Alice's portion of the message and b) appends Alice's ID to the Bob's portion of the message before the encryption.

What did we achieve with this? First, for each session key, the server does only two encryptions. Because Alice is the only one who can decrypt her part of the message, she knows that this session key is for her. Also because her part includes Bob's identifier, she can be sure that another encryption of the session key was encrypted using Bob's long-term key. The same holds for Bob. In this case, Carol cannot simply intercept and change messages in between Alice and the server. This version of the protocol looks even better and protects against more attacks.

Slide 27. Let's add one more assumption about the attacker. Let's assume that the past session keys might be leaked in some way. For example, a user may accidentally publish it somewhere on the Web, or his computer might have been compromised, or maybe it was used insecurely in other higher level protocols.

How can it be used by Carol? Assume that at some point in the past Alice used our protocol to exchange a shared key with Bob and Carol (the attacker) was able to intercept and save all protocol messages during that past run. Later on the shared key was leaked. (Note that at this point Alice and Bob had already finished their session and don't use this session key anymore.)

At this point Carol has the old shared key from the past, and also the past protocol messages encrypted with this key. Now assume that Alice wants to initiate a new communication with Bob and initiates a new protocol urn. Carol is able to intercept Alice's new request to the server and replaces the server's response with the message from the older protocol run (this old message contains the old leaked key). Alice continues the protocol and sends that encrypted key to Bob. In this way Alice and Bob will end up negotiating the session key that was used in the past (i.e. the one that was leaked). At this point Carol, is able to decrypt and read Alice's and Bob's communications.

Slide 28. Why did it happen? Because our threat model did not include the possibility of leaking session keyfs used in the past. Let's redesign our protocol. One of the ways to protect from such replay attacks is to incorporate new random values for each new protocol run. This random value is called "nonce" which stand for "Number used ONCE". Let's see how we can improve our protocol by using a nonce.

When Alice contacts the server, she generates a nonce and sends it along with her and Bob's identifiers. The server then generates a new session key and constructs the reply to Alice as follows. It first encrypts the session key and Bob's identifier with Bob's long term key. It then appends the nonce, the session key, and Bob's ID, and encrypts it with Alice's long term key. It then sends this reply to Alice.

Alice decrypts the reply and verifies that it contains the same Nonce as the one she sent. Since the Nonce was newly generated for this specific session, she knows that the server's reply is a part of the new protocol run and should contain a new session key. She then continues the protocol and sends Bob's part of the message to Bob. At this point, Bob might also verify that Alice knows the key by sending another random number back to Alice (encrypted with the new session key). Alice can to decrypt it with the new session key, compute Nonce+1 and send it (encrypted) back to Bob.

It seems we solved the problem with key freshness for Alice. But what about Bob, can we still mount a replay attack?

Slide 29. Assume there was a previous session key, K', that was leaked at some point. In this case, if the attacker Carol can intercept protocol messages sent to Bob and modify them, then she can mount a similar attack that we considered before.

If Carol intercepted and recorded the encrypted protocol messages for this past leaked session key, she can replay them to Bob during the third step of the protocol. At this stage, the protocol does not provide any information whether it's a new session key or it was used before. During steps 4 and 5, even though Bob encrypts a Nonce, he encrypts it with the old session key. These messages can be intercepted by Carol who can successfully decrypt them, apply the needed transformation and reply to Bob. In this way, Bob will think that he communicates with Alice with a newly generated session key.

Slide 30. Let's once again try to improve the protocol. In order to solve the previous problem, Bob needs to generate his Nonce that would be used by the server. Let's add additional step at the beginning of the protocol. During this step, Bob generates a nonce N_B, and sends it to Alice along with his ID. Alice then forwards it to the server along with her own Nonce N_A.

The server then generates a new session key K_{AB} and prepares the reply consisting of two parts. The first part is for Alice and includes Alice's Nonce encrypted with Alice's long-term key. The second part is for Bob and includes his Nonce and is encrypted with Bob's long-term key. During step 4, Alice sends this part to Bob. Bob then decrypts it and verifies the Nonce. In this way, he can see that this is a newly generated session key. This version of the protocol look much better now.

Slide 31. What we were trying to do so far is one of the common ways to construct a cryptographic protocol. With this approach, we start with a security goals. Then we the need to think about the corresponding threat model, including assumptions about the attacker's capabilities. This process is iterative, and with each iteration we add new assumption/attacks and introduce appropriate protection mechanisms.

In our example, we started with 3 security goals and designed a minimal protocol that was able to achieve some of them. We then introduced a threat model and gradually extended it by adding more attacks. At the same, with every new iteration of the threat model, we also re-designed our protocol against these new attacks. And little by little we came up with something more useful.

How do we know that our last version of the protocol is safe? The answer is we don't know. We can try to use automated formal techniques and tools. A few examples include ProVerif, Tamrin, and AVISPA. Or we can carefully analyze the protocol against existing attacks. However none of these techniques is perfect.

When you finished designing a protocol and already audited it yourself, it is a good idea to publish it and ask for public review. Public feedback can be very useful; and relying on public review is a common practice for new cryptographic protocols and algorithms.

Finally there is test of time. Many protocols are found to be flawed after many years since they were published and deployed including TLS protocol. This is not always bad because discovering new flaws helps us design better protocols.

Slide 32. A few words about freshness. Freshness critical in preventing replay attacks. In general, there two ways to get freshness guarantee for some values, for example for a session key.

In the first approach, the user takes part in generating that value (or the key). For example, if the protocol involves two users, each of them can generate a fresh nonce: N_A and N_B These two nonces then can be passed as arguments to function 'f' that generates the session key. Function 'f' obviously should be chosen so that old keys could not be generated even if N_A and N_B are known. In order to achieve this property, we need to use some information that is never transmitted unencrypted. For example it can be a master key, or some random data.

Slide 33. In the second approach, a user can rely on some other piece of information received along with the value. This additional information should be known to be fresh. For example it can be a timestamp, a nonce, or a counters that increase with every iteration of the protocol.