CS 261 Scribe Notes
Crypto Protocols

Instructor: *Prof. David Wagner*
Scribe: *Mayank*

April 16, 2021

1. Designing Cryptographic Protocols

Back in the 90's, there was a lot of research focus on designing secure cryptographic protocols. However, many of these proposals were rife with security flaws, even when the proposed algorithms were as simple as five lines of interactions between principals. To address this problem, in 1996, Martin Abadi and Roger Needham decide to write a paper [1] outlining fundamental principles whose adherence would potentially lead to more robust protocols avoiding the most common failures.

A natural question to ask here is how relevant are the principles laid out by them in [1] today. It is quite reasonable to believe that these principles aren't as relevant today as they were a few years ago; we use standard well-vetted protocols today and rarely design our own cryptographic protocols from scratch. In addition, unlike the 90's when the only available building blocks in designing protocols were encryption and signatures, we have more powerful primitives today which make it much easier to design robust protocols. Some of these primitives include:

1. Point-to-point secure channels. Standard protocols like SSL can be used for this.
2. Sealed data or authenticated encryption for securely storing data.

2. Notation

We use $A \rightarrow B : m$ to mean that A sends the message m to B, where A, B are principals (a.k.a parties) involved in the interaction. $\{m\}_K$ denotes that m is encrypted with the key K. $[m]_{K^{-1}}$ denotes that m is signed with the key K^{-1}. The letter K is reserved for key, T for timestamp and N for nonce (number used only once). Any subscripts to the letters usually refer to the principal who owns/generates that key/nonce. While describing the protocols, the intended protocol interaction is numbered from $1 \ldots n$, while an interaction involving an attack is numbered from $1' \ldots n'$, where n (or n') is the total interactions in the protocol. Unless otherwise stated, we use the word "message" to refer to any interaction that happens between two principals.

3. Flaws in Protocols

We now look at several protocols, some made-up, others published, and look at flaws and potential mitigations to address them.

3.1 Warmup

We start with a straw-man protocol first. The protocol intends to establish a secure communication channel (*authenticated* as well as *encrypted*) between Alice (A) and Bob (B) by using the so-called "challenge-response"-style protocol. Assume that all principals know the public keys of all other principals.

Warmup Protocol

1. $A \rightarrow B: \quad A$

2. $B \rightarrow A: \quad N_B$

3. $A \rightarrow B: \quad [N_B]_{K_A^{-1}}$

4. $A \rightarrow B: \quad \{\text{message}\}_{K_B}$

5. $A \rightarrow B: \quad \{\text{message'}\}_{K_B}$

Analysis: We want the communication channel to be both authenticated and confidential, meaning that in steps **4.**, **5.**, only A can send the message to B and no one else can see the contents of the message. Confidentiality is easy to see: since the messages are encrypted with K_B, only B can successfully decrypt and read the messages. However, authenticity of the true origin of the message isn't preserved here. More concretely, a man-in-the-middle can take messages from $A \rightarrow B$, discard them, encrypt new custom messages (steps **4'.**, **5'.** in the figure below) with K_B and forward them to B. There is no way in this protocol for B to know whether message came from A or the attacker.

The reason that the attack succeeds is that the steps **1.**, **2.**, **3.** were used to prove to B that A is present online, but steps **4.**, **5.** are not bound to that proof in any way. So, anyone can claim to be A when they send a message to B.

Breaking Warmup Protocol

M is the man-in-the-middle

1. $A \rightarrow B:\quad A$

2. $B \rightarrow A:\quad N_B$

3. $A \rightarrow B:\quad [N_B]_{K_A^{-1}}$

4'. $M \rightarrow B:\quad \{\text{evil}\}_{K_B}$ // M changes "message" to "evil"

5'. $M \rightarrow B:\quad \{\text{evil'}\}_{K_B}$ // M changes "message'" to "evil'"

3.2 X.509 standard #1

We now shift our focus to published protocols. Similar to the previous protocol, we want confidentiality and authenticity of the communication that happens from $A \rightarrow B$.

X.509 standard #1

1. $A \rightarrow B:\quad A, [T_A, B, \{\text{message}\}_{K_B}]_{K_A^{-1}}$

Early Analysis: Notice that in the protocol above, the signature on the message $(T_A, B, \{\text{message}\}_{K_B})$ in step **1.** is generated by A meaning that she approves of its contents. However, the $\{\text{message}\}_{K_B}$ component of the message is a ciphertext and there is no reason to believe that A is actually aware of what information it encrypts. In other words, A is able to endorse something that she doesn't necessarily know. We now see how this leads to an attack.

Example 1 (Remote Log-in): Let's look at a particular use-case of this protocol. Suppose A wants to log-in to a remote server B by providing the access password. A doesn't want to reveal the password to an eavesdropper, so she encrypts it with the public key K_B of server B and signs it with her private key K_A^{-1} to prove her identity. Server B first checks that A's signature is valid, decrypts the password and grants access to A if password is correct.

Remote log-in using X.509 standard #1

1. $A \rightarrow B:\quad A, [T_A, B, \{\text{password}\}_{K_B}]_{K_A^{-1}}$

Analysis: An eavesdropper M can intercept A's log-in request from step **1.**, extract $\{\text{password}\}_{K_B}$ from it and then use it to curate a valid log-in request to server B *without* *ever knowing the password* (see step **1'.**).

Breaking X.509 standard #1

1. $A \to B:\quad A, [T_A, B, \{\text{password}\}_{K_B}]_{K_A^{-1}}$ // M intercepts $\{\text{password}\}_{K_B}$

1'. $M \to B:\quad M, [T_M, B, \{\text{password}\}_{K_B}]_{K_M^{-1}}$ // M gets access

Example 2 (Secure Auctions): Another use-case is closed-price auctions — where the auctioneer B collects an encrypted bid from all participants and after the deadline passes, he opens all the bids to decide the winner. Similar to the previous example, a participant M can reuse encrypted bid submitted by A to construct a valid bid matching that of A without knowing what A's bid was.

It is debatable whether the attacks shown above in the two examples are actual security flaws of X.509 standard #1, or are they just a consequence of using the standard for an unsuitable use-case.

Lessons Learned: We must understand that authentication can be seen to serve two purposes in different situations: a) the signer holds responsibility of whatever is being signed (or endorsement), or b) the signer uses the signed message as a way of proving knowledge of certain information (or claiming credit). While encrypting before signing is a secure way to achieve a), it is not enough for b). If we want to prevent the attacks shown above, we can simply tweak the protocol to require the clients first sign and then encrypt the password, i.e. $\{[\text{ password }]_{K_A^{-1}}\}_{K_B}$.

3.3 Telecommunications Management Network (TMN)

Assume A and B have the public key K_S of server S which provides a way for them to quickly establish a shared symmetric key k_B between themselves. Both A and B send their keys k_A and k_B, respectively, to S by encrypting them with K_S to prevent leaking them to an eavesdropper. The server responds by sending $k_A \oplus k_B$ to A, where $\oplus$ is the bitwise XOR operation. A can recover k_B by doing $k_A \oplus (k_A \oplus k_B)$. Now both A and B know the shared key k_B, as desired. This protocol, called TMN [2], was used for resource-constrained (can do encryption efficiently, but not decryption) mobile devices A and B to establish a shared key between themselves. We want that no other party can learn k_B except for A, B and S.

TMN [2]

1. $A \to S:$ $\{k_A\}_{K_S}$

2. $B \to S:$ $\{k_B\}_{K_S}$

3. $S \to A:$ $k_A \oplus k_B$ // A recovers k_B as $k_A \oplus (k_A \oplus k_B)$

Analysis (Abusing Oracles): While a subtle man-in-the-middle attack can be used by an attacker to learn k_B and therefore see all messages $B \to A$, a pair of attackers M, M' can launch a much more powerful attack here by abusing the server S as an oracle to decrypt any key k_B of their choice. In particular, in step **1'.**, M plays the ciphertext that he wants S to decrypt. This is followed in step **2'.** by M' submitting a genuine query to S which he has already disclosed to M. From the response that M receives in step **3'.** and the query that M' submitted, M can recover the correct decryption of the ciphertext by $k_{M'} \oplus (k_{M'} \oplus k_B)$.

Breaking TMN

1'. $M \to S:$ $\{k_B\}_{K_S}$ // M uses intercepted encrypted key $\{k_B\}_{K_S}$

2'. $M' \to S:$ $\{k_{M'}\}_{K_S}$

3'. $S \to M:$ $k_B \oplus k_{M'}$ // M, M' can recover k_B

3.4 Needham-Schroeder

Needham-Schroeder (NS) [3] is quite widely known as "grand daddy of flawed protocols" and was intended for A and B to establish a secure channel (authentication, confidentiality and timeliness) using each other's public keys. This was proposed in 1978 and became one of the standard protocols for setting up secure channels at that time, until Gavin Lowe in 1996 found an attack [4] against this protocol. This led to design of the well-known and widely-used Kerberos [5] protocol where these attacks were mitigated. The NS protocol is very straightforward. A and B send challenges and responses using fresh nonces N_A and N_B in steps **1.** and **2.**. The final message **3.** serves as the response to B's challenge as well as encryption of the message which was to be sent to B.

Needham-Schroeder [3]

1. $A \to B: \quad \{A, N_A\}_{K_B}$

2. $B \to A: \quad \{N_A, N_B\}_{K_A}$

3. $A \to B: \quad \{N_B, \text{message}\}_{K_B}$

Analysis (The Lowe Attack [4]): Let us consider that Dave (D) is a man-in-the-middle who wants to send a custom message to B while claiming to be A. The only precondition for the attack to successfully work is that A initiates a connection request with D (step **1.**). After getting this request, in step **1'.**, D decrypts it, encrypts it with K_B and forwards it over to B claiming to be A. Getting a request from D (claiming to be A), B sends a challenge in step **2'.** to A asking for them to prove that they are indeed A. However, since the nonce N_A inside this message matches with the challenge that A sent to D in step **1.** and due to the lack of identification that the message is actually coming from B, A believes that the message in step **2'.** came from D. Therefore, A replies to the challenge to D in step **3.**, effectively getting abused by D into decrypting the challenge that B asked for. D can now learn N_B and use it as a proof to send any message to B claiming to be A (step **3'.**). A potential mitigation for this attack is to add the identity of the challenger in step **2'.**, i.e. it becomes $B \to A: \quad \{N_A, N_B, B\}_{K_A}$. Or we could ask the challenger to sign N_A in step **2'.** which can then be used by A to be sure of the identity of the challenger.

Breaking Needham-Schroeder: The Lowe Attack [4]

1. $A \to D: \quad \{A, N_A\}_{K_D}$

Dave can convince Bob that he is Alice:

1'. $D \to B: \quad \{A, N_A\}_{K_B}$

2'. $B \to A: \quad \{N_A, N_B\}_{K_A}$ // A thinks this came from D ($\because$ of N_A)

3. $A \to D: \quad \{N_B, \text{message}\}_{K_D}$ // D abused A as oracle

3'. $D \to B: \quad \{N_B, \text{message'}\}_{K_B}$

3.5 Systems Management Architecture for Server Hardware (SMASH)

Similar to previous protocol, A and B want to establish a secure channel given the knowledge of each other's public keys. In cryptographic protocols, nonces are used as a way of proving freshness. Nonce are usually unpredictable (we will see what happens when this property isn't satisfied in section 4.) and non-repeating values so that they cannot be replayed and help enforce active presence of the person one is talking to. The protocol below is quite simple to understand.

The Smash Protocol [6]

1. $A \to B:$ $\{A, N_A\}_{K_B}$

2. $B \to A:$ $\{N_A\}_{K_A}$

3. $A \to B:$ $\{N_A, \text{message}\}_{K_B}$

Analysis: Observe that something funny is going on in the protocol above. B is not providing a nonce challenge in the protocol. Skipping this challenge is the perfect recipe for replay attacks. In other words, B is proving to A that he is online, but never checking if A is online or not. Let us consider an attacker Zorro (Z) who wants to impersonate A when communicating with B. In step **2.** above, B was expecting that no one other than A would be able to decrypt and learn N_A. However, the person who created the nonce knows it already and then the message sent by B in step **2.** serves no purpose of ensuring any identity-related information of who sent the nonce. Zorro uses this exact same fact to send a random nonce in step **1'.** claiming to be A. The message from B in step **2'.** serves no purpose for Z and he disregards it (assume that A is offline too). Finally, in step **3'.**, since Z knew N_Z already, he can just successfully send any message to B claiming to be A by using N_Z as a proof and B thinks that since N_Z could only be decrypted by A, it is a valid proof (overlooking that the creator Z knows it too). As a mitigation, B should include a fresh challenge in the message in step **2.** and verify its response in step **3.**.

"Smashing" The Smash Protocol

1'. $Z \to B:$ $\{A, N_Z\}_{K_B}$ // Z impersonates A

2'. $B \to A:$ $\{N_Z\}_{K_A}$ // For B, this serves no purpose

3'. $Z \to B:$ $\{N_Z, \text{message}\}_{K_B}$ // B thinks it is from A

3.6 Online Shopping

This is a protocol that was actually used in real-world for third-party payments in online shopping; unlike the previous protocols, this is more relevant today. In this protocol, a customer C buys something from the store website S and makes the payment for that transaction via PayPal P. All the interactions between the principals involved here happens through URLs; the customer can be thought of as using a web browser which visits the required URLs in order as the protocol proceeds. In step **1.**, customer clicks the buy button on the store website and is redirected by the website to a transaction-specific (with unique transaction identifier `id`, transaction amount `amount` and the beneficiary `to` which happens to be the store S here) URL on PayPal in step **2.**. C authorizes the payment in step **3.**, is redirected back to the store website on a post-transaction landing page in steps **4.** and **5.** with a PayPal transaction identifier `tx`. As soon as C reaches the landing page, S checks with P using `tx` if the transaction was completed through step **6.**, followed by a response from P listing the store-specific transaction `id` and `amount` of what was authorized by C (back in step **3.**). If this information matches the expectation, the protocol is complete and S updates the status to C.

A Shopping Protocol

C = Customer, S = Store and P = PayPal

1. $C \to S:$ `/buy`

2. $S \to C:$ *redirect to:* `P/pay?id=i&amount=p&to=S&u=S/done`

3. $C \to P:$ `/pay?id=i&amount=p&to=S&u=S/done`

4. $P \to C:$ *redirect to:* `S/done?tx=t`

5. $C \to S:$ `/done?tx=t`

6. $S \to P:$ `/check?tx=t`

7. $P \to S:$ `id=i, amount=p`

8. $S \to C:$ Goods are on the way

Analysis: The starting point for us to find an attack here is the difference between the information used by C to make the payment in step **3.**, i.e. (`id`, `amount`, beneficiary `to`) vs. what S gets to know from P post-transaction in step **7.**, i.e. only (`id`, `amount`). Since S never gets to know who the actual beneficiary in the attempted transaction was, it might be thinking that money got transferred to it while the actual transfer was made to someone else. This is the attack. C can change the `to` field in the URL (as shown in step **3'.** below) and make the payment to themselves, while S it oblivious to this cheating. Hence, C retained their bank balance and essentially convinced S into believing that S was paid. An easy fix is to include the beneficiary `to` in step **7.**, so that S can check if they "really" got paid.

Shopping for "Free"

C = Customer, S = Store and P = PayPal

1. $C \to S$: `/buy`

2. $S \to C$: *redirect to:* `P/pay?id=i&amount=p&to=S&u=S/done`

3'. $C \to P$: `/pay?id=i&amount=p&to=C&u=S/done` // C changes `to` to C

4. $P \to C$: *redirect to:* `S/done?tx=t`

5. $C \to S$: `/done?tx=t`

6. $S \to P$: `/check?tx=t`

7. $P \to S$: `id=i, amount=p`

8. $S \to C$: *Free* goods are on the way

4. Discussion by Stephan

Stephan started the discussion with vulnerabilities that existed in the chip and pin cards from EMV (Europay, Mastercard and Visa) when they were first announced to counter the ever-growing mag-strip related card counterfeiting [7]. The idea behind these cards was to have the chip as a tamper-resistant store of secret information to prevent counterfeiting (extracting this information physically damages the chip) and the pin prevents the card-use when it gets stolen. However, this led to an increase in "card-not-present" transactions — which included internet, mail-order and phone-based transactions — for committing fraud (as shown in figure 1), and mag-strip was still supported at most places as the fallback option during this transition period.

The new protocol for verifying card transaction had three steps as shown in figure 2:

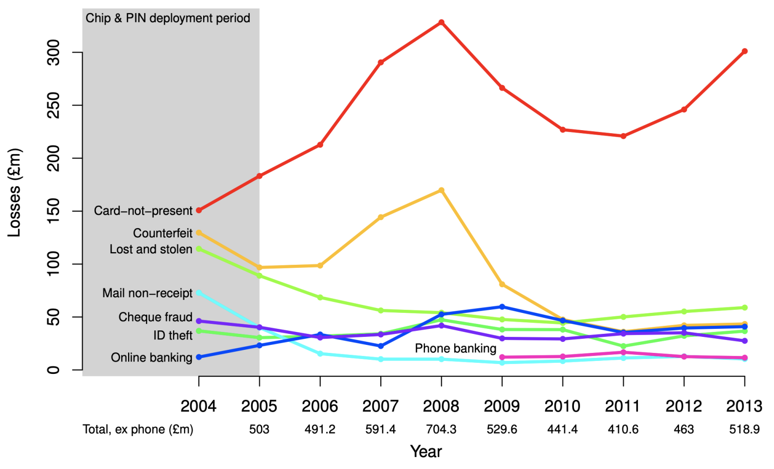
1. *Card Authentication*: Card details read and authenticated by an ATM or merchant terminal. In this step, the card provides the ATM with data records containing fields like card number, expiry date and protocol options. In order to prevent the attackers from using known/guessed card numbers, the card is required to sign these records using an RSA signature for the ATM to verify.

2. *Cardholder Verification*: The person using the card proves that they are legitimate cardholder by presenting pin or signature.

3. *Transaction Authorization*: The issuing bank decides if transaction should proceed or not. This step involves a series of interactions between the issuer, ATM and card.

It starts with the ATM generating an unpredictable nonce (this is crucial; see section 4.1) and sending T = (amount, currency, date, . . ., nonce, . . .) to the card. The card responds with a Authorization Request Cryptogram (ARQC) which is essentially a Message Authentication Code (MAC) on T along with card-specific data like Application Transaction Counter (ATC) which is a 16-bit counter incremented with each transaction. In the next step, ARQC and the encrypted pin are sent from the card to the issuer through the ATM. The issuer verifies the pin and the MAC inside ARQC and sends back Authorization Response Cryptogram (ARPC) with an Authorization Response Code (ARC) to the ATM, which is forwarded to the card. ARPC is typically MAC(ARQC) $\oplus$ ARC. For the ATM, ARC serves as the green-signal to dispense the money. Finally, the card computes Transaction Certificate (TC) by computing a MAC on the whole transcript so far. TC is used to settle the transaction later.

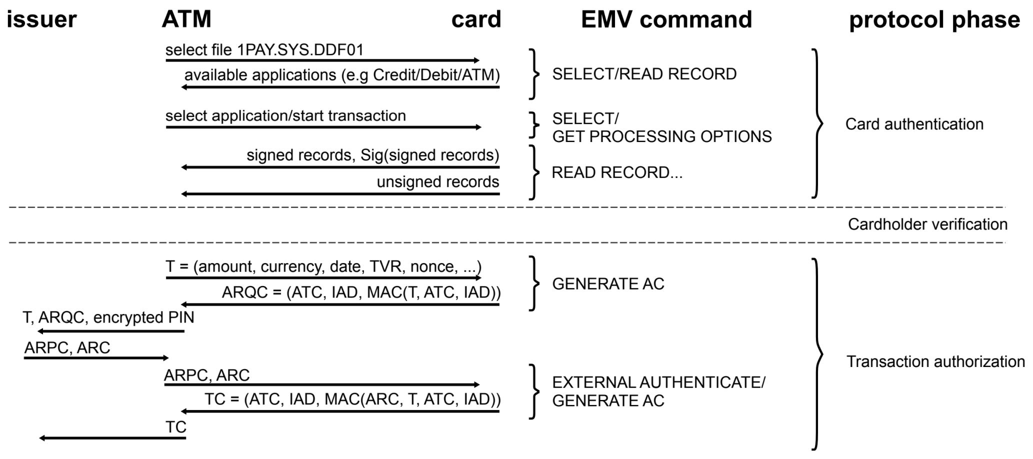
Figure 1: Fraud levels on UK-issued payments cards



4.1 The Pre-play Attack

Let's assume that the nonce generated by the ATM is a predictable number, what can go wrong if that is the case? Attackers can use fraudulent ATM machines/merchant terminals to have a card generate ARQC for the nonce of their choice. They can then use that ARQC to fool an innocent ATM machine into believing that it is talking to a real card, while the attackers just replay the pre-recorded ARQC message. If the attackers are able to successfully predict the nonce that an innocent ATM produces, they can pre-record an ARQC response from a valid card and use that against the innocent ATM. The ATM has no way of knowing that the ARQC is pre-recorded as nonce was the entity that guaranteed freshness and to the issuer, this is a legitimate ARQC too.

Figure 2: Outline of an EMV transaction at ATM


issuer
ATM
card
EMV command
protocol phase
select file 1PAY.SYS.DDF01
available applications (e.g Credit/Debit/ATM)
SELECT/READ RECORD
select application/start transaction
SELECT/
GET PROCESSING OPTIONS
signed records, Sig(signed records)
unsigned records
READ RECORD...
Card authentication
Cardholder verification
T = (amount, currency, date, TVR, nonce, ...)
ARQC = (ATC, IAD, MAC(T, ATC, IAD))
GENERATE AC
T, ARQC, encrypted PIN
ARPC, ARC
ARPC, ARC
TC = (ATC, IAD, MAC(ARC, T, ATC, IAD))
EXTERNAL AUTHENTICATE/
GENERATE AC
TC
Transaction authorization

The card industry didn't specify what it meant for a nonce to be unpredictable, which led to implementations where counters, weak RNGs (Random Number Generators), RNG seeds with little to no entropy, etc. were used for nonce (which obviously are quite predictable) and that set the necessary ground for this attack to rise.

The attack works as follows:

1. Find a bad RNG.
2. Harvest ARQC and TC for a bunch of nonce values from a victim card.
3. Use them to cash out of honest ATMs.

References

[1] Martín Abadi and Roger M. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Software Eng.*, 22(1):6–15, 1996.

[2] ITU-T Recommendation M.3010. Principles for a telecommunications management network. https://www.itu.int/rec/T-REC-M.3010, 1996.

[3] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[4] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.

[5] John T. Kohl and B. Clifford Neuman. The kerberos network authentication service (V5). *RFC*, 1510:1–112, 1993.

[6] Distributed Management Task Force. Systems Management Architecture for Server Hardware (SMASH) Command Line Protocol (CLP) Architecture White Paper. https://www.dmtf.org/sites/default/files/standards/documents/DSP2001_1.0.1.pdf, 2006.

[7] Mike Bond, Omar Choudary, Steven J. Murdoch, Sergei P. Skorobogatov, and Ross J. Anderson. Chip and skim: Cloning EMV cards with the pre-play attack. In *IEEE Symposium on Security and Privacy*, pages 49–64. IEEE Computer Society, 2014.