BTI 4201: From Symmetric Encryption to Secure Channels

Christian Grothoff

Berner Fachhochschule

26.5.2024

Learning Objectives

Review: Security Games

Example: Attack on CBC Stateful IV

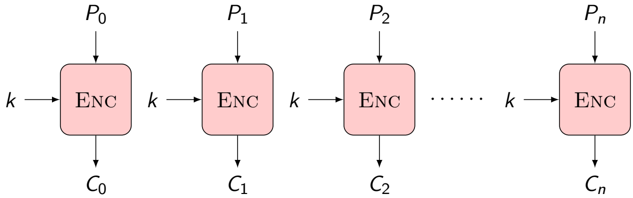
Beyond IND-CPA

Real-world use of cryptographic primitives (exercise)

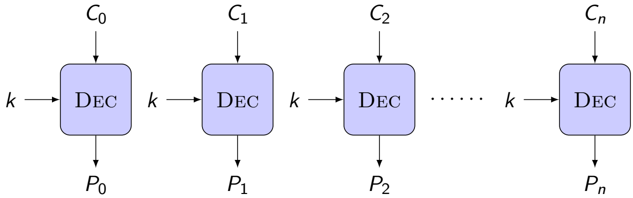
Symmetric key establishment protocols

Secure channels

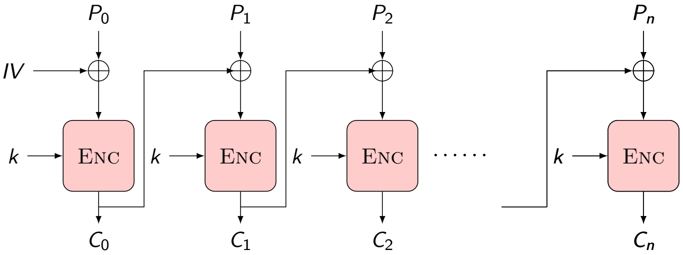
ECB encryption



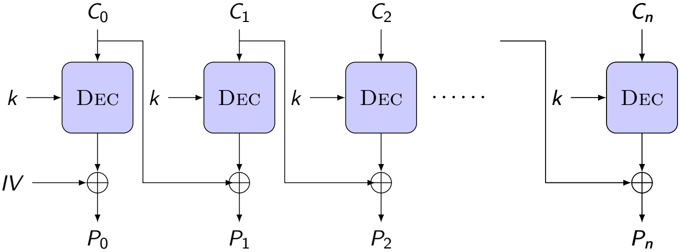
ECB decryption



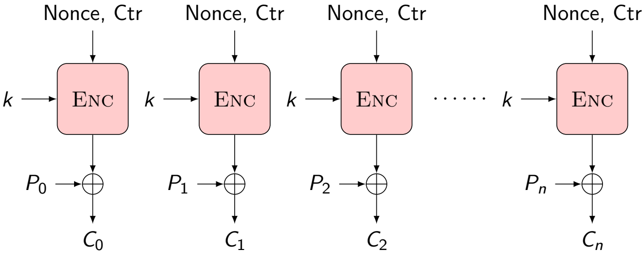
CBC encryption



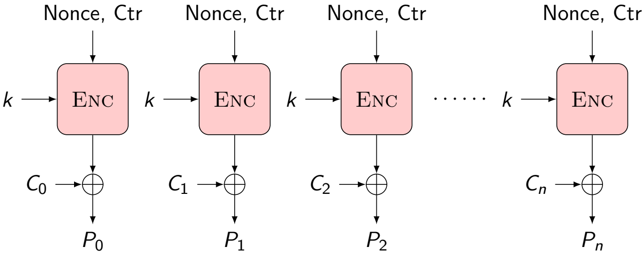
CBC decryption



CTR encryption



CTR decryption



Problem

Which mode is secure?

Problem

Which mode is secure?

How to prove it?

Security Definitions for Symmetric Encryption

Simplistic security definitions would be:

1. It must be impossible for an adversary to find the key from ciphertexts.
2. It must be impossible for an adversary to find the plaintext from a ciphertext.

Security Definitions for Symmetric Encryption

Simplistic security definitions would be:

1. It must be impossible for an adversary to find the key from ciphertexts.
2. It must be impossible for an adversary to find the plaintext from a ciphertext.

These are insufficient as, for example, they do not capture the insecurity of the ECB mode!

Problem

We need a precise, succinct and comprehensive security definition!

Subtle Corner Cases

Given n stocks, the message $m := m_1||m_2||m3||\ldots||m_n$ tells your broker to buy i-th stock if $m_i = 1$ or to sell if $m_i = 0$. Suppose m is encrypted and sent to your broker. We would consider the encryption to have failed if an adversary can even just compute *one bit* of the message to learn whether you want to buy or sell stock i.

Subtle Corner Cases

Given n stocks, the message $m := m_1||m_2||m3||\ldots||m_n$ tells your broker to buy i-th stock if $m_i = 1$ or to sell if $m_i = 0$. Suppose m is encrypted and sent to your broker. We would consider the encryption to have failed if an adversary can even just compute *one bit* of the message to learn whether you want to buy or sell stock i.

Even partial information leakage about a message is problematic.

Subtle Corner Cases

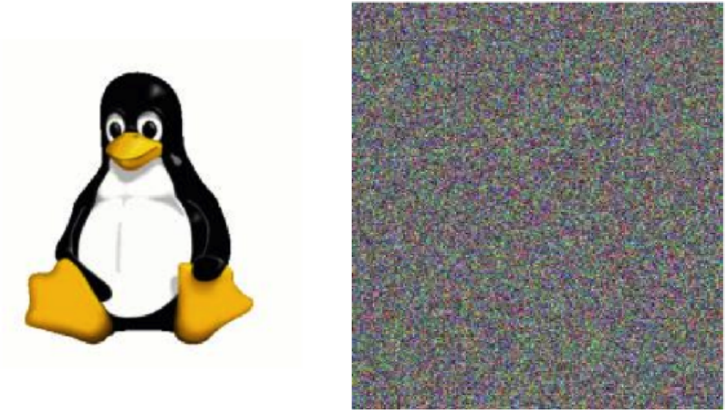
Given n stocks, the message $m := m_1 || m_2 || m3 || \ldots || m_n$ tells your broker to buy i-th stock if $m_i = 1$ or to sell if $m_i = 0$. Suppose m is encrypted and sent to your broker. We would consider the encryption to have failed if an adversary can even just compute *one bit* of the message to learn whether you want to buy or sell stock i.

Even partial information leakage about a message is problematic.

In fact, even *probabilistic* leakage is a problem: an adversary that can tell that with probability of 90% whether you are buying or selling might be a problem!

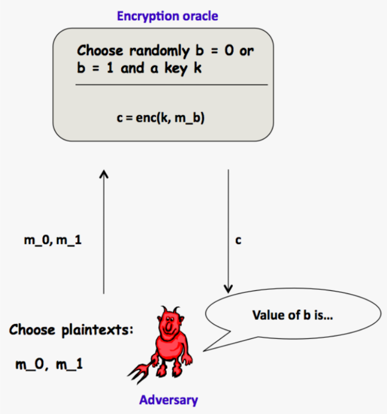
What we want

Our goal is to formalize the intuitive notion of secure encryption shown here:



The picture shows that an adversary does not learn any useful information about a plaintext from a ciphertext.

Indistinguishability under Chosen Plaintext Attacks (IND-CPA)



Encryption oracle

Choose randomly b = 0 or b = 1 and a key k

c = enc(k, m_b)

m_0, m_1

c

Value of b is...

Choose plaintexts:

m_0, m_1

Adversary

Indistinguishability under Chosen Plaintext Attacks (IND-CPA)

Security Game: Adversary chooses m_1 and m_2. Defender chooses key k and $b \in \{0, 1\}$. Defender computes $c := \texttt{enc}(k, m_b)$ and gives c to the adversary.

Definition: A symmetric encryption scheme `enc()` is *IND-CPA secure*, if it is impossible for all possible adversaries to tell whether $b = 0$ or $b = 1$. That is, the adversary wins if they can determine the correct b.

Problem

The above definition is incomplete: What if the adversary wins 60% of the time?

Cryptographic Games

An *oracle* is a party in a game that the adversary can call upon to indirectly access information that is otherwise hidden from it. **IND-CPA** can then be formalized like this:

Setup Generate random key k, select $b \in \{0, 1\}$ for $i \in \{1, \ldots, q\}$.

Oracle Given M_0 and M_1 (of same length), return $C := \texttt{enc}(k, M_b)$.

The adversary wins, if it can guess b with probability greater than $\frac{1}{2} + \epsilon(\kappa)$ where $\epsilon(\kappa)$ is a negligible function in the security parameter κ.

Restrictions on Oracle use

Many schemes break after an large number of messages. Thus, restrictions are generally imposed on the use of the Oracle by the adversary:

- Best known attack on AES uses birthday attack, 2^{64} queries

⇒ limit oracle use to say 2^{30} queries of some maximum length, say 2^{13} (1 kB).

Then the resulting *advantage* of the adversary remains “small”.

IND-CPA

IND-CPA is a widely accepted definition of secure symmetric encryption.

Practically relevant symmetric encryption schemes (i.e. AES in CTR or CBC mode) are considered IND-CPA secure.

Examples for IND-CPA Insecure Schemes

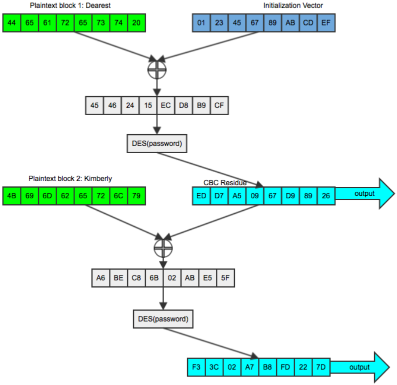
- Schemes where the plaintext can be recovered from the ciphertext ...
- Schemes where the key can be recovered from the ciphertext ...
- ECB mode encryption ...
- Schemes where the n-th plaintext bit can be recovered from ciphertext ...

... are all IND-CPA insecure.

Examples for IND-CPA Insecure Schemes

- Any deterministic, stateless encryption scheme is insecure.
- CBC stateful IV mode (where IV is *predictable* because, for example, sender determines next IV by incrementing previous IV) is IND-CPA insecure

Attacking CBC stateful IV (1/5)[1]



Plaintext block 1: Dearest

44 65 61 72 65 73 74 20

Initialization Vector

01 23 45 67 89 AB CD EF

45 46 24 15 EC D8 B9 CF

DES(password)

Plaintext block 2: Kimberly

4B 69 6D 62 65 72 6C 79

CBC Residue

ED D7 A5 09 67 D9 89 26

output

A6 BE C8 6B 02 AB E5 5F

DES(password)

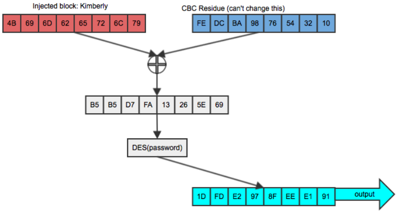
F3 3C 02 A7 B8 FD 22 7D

output

Goal: confirm “Kimberly” was sent!

Attacking CBC stateful IV (2/5)

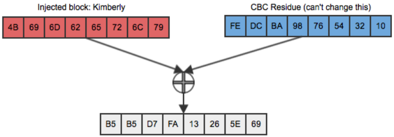
Setup: Get oracle to encrypt “Kimberly”:



Given random CBC residue, this does not help.

Attacking CBC stateful IV (3/5)

CBC residue is XORed with input, get rid of it first using *predicted* IV:



Injected block: Kimberly

4B 69 6D 62 65 72 6C 79

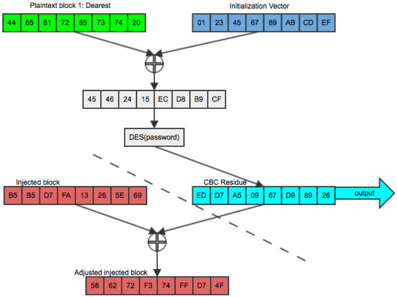
CBC Residue (can't change this)

FE DC BA 98 76 54 32 10

B5 B5 D7 FA 13 26 5E 69

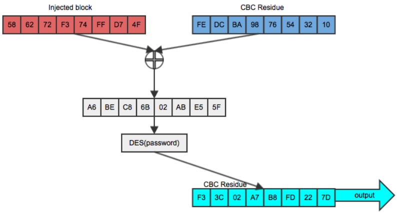
Attacking CBC stateful IV (4/5)

Then add the residue from the original encryption:



Attacking CBC stateful IV (5/5)

Now confirm the output matches:



Injected block

58 62 72 F3 74 FF D7 4F

CBC Residue

FE DC BA 98 76 54 32 10

A6 BE C8 6B 02 AB E5 5F

DES(password)

CBC Residue

F3 3C 02 A7 B8 FD 22 7D

output

If output matches, original text was “Kimberly”.

Summary

For CBC, if an attacker can:

- guess the plaintext corresponding to any ciphertext block they have seen before, and
- can predict a future IV, and
- can submit a suitable message to be encrypted with that IV,

then they can verify their guess.

Is this attack an issue?

- Requires guessing the entire block
- Requires access to encryption oracle
- Block size is say 8 bytes, so 2^{256} trials

Is this attack an issue?

- Requires guessing the entire block
- Requires access to encryption oracle
- Block size is say 8 bytes, so 2^{256} trials

BEAST (2011) made this attack practical by shifting each unknown plaintext byte to a position in the block just after 7 bytes of known plaintext.

IND-CPA Secure Schemes

- The CTR random IV symmetric encryption scheme is IND-CPA secure.
- The CTR stateful IV encyption scheme (ensuring no IV re-use) is IND-CPA secure.
- The CBC *random* IV symmetric encryption scheme is IND-CPA secure.

All of the above **assume** that the underlying cipher is (indistinguishable from) a PRF.

Pseudo random functions (PRF)

- A *pseudo random function (PRF)* is a function that is (computationally) indistinguishable from a true random function
- The previous positive results are true under the *assumption* that the block cipher used (e.g. AES) is a PRF.
- Assumption really means that this is a commonly shared belief of the crypto community. No proof exists!
- Breaking any of these schemes thus means breaking the PRF property of the underlying block cipher.

The crucial security property of a secure block cipher is that it is (indistinguishable from) a PRF!

Part II: Chosen Ciphertext Attacks

IND-CPA vs. Chosen Ciphertext

IND-CPA is **not** the strongest security model!

- The adversary does not have access to a *decryption* oracle
- With a decryption oracle, an adversary can be allowed to ask for *some* messages of its choice to be decrypted.
- Security is achieved only if *other* messages still remain indistinguishable.

Indistinguishability under Chosen Ciphertext Attacks (IND-CCA)

The adversary's goal is the same as in IND-CPA (determine b given $\mathtt{enc}(k, M_b^i)$) for sequences of messages $M_{0,1}^i$).

Setup Generate random key k, select $b \in \{0, 1\}$.

Oracle E Given M, return $C := \mathtt{enc}(k, M)$.

Oracle D Given C', return $M := \mathtt{dec}(k, C')$.

The additional restriction $C' \neq C$ must be imposed on the use of Oracle D: The adversary is not allowed to ask for decryption of a ciphertext C that was previously returned by the encryption oracle.

Examples for IND-CCA Insecure Schemes

CTR schemes are IND-CCA insecure:

> *"Say $\langle r, C \rangle$ is a ciphertext of some l-bit message M, and we flip bit i of C, resulting in a new ciphertext $\langle r, C' \rangle$. Let M' be the message obtained by decrypting the new ciphertext. Then M' equals M with the i-th bit flipped. Thus, by making a decryption oracle query of$\langle r, C' \rangle$one can learn M' and thus M."*
>
> *–Symmetric Encryption by Mihir Bellare and Phillip Rogaway*

Problem

IND-CCA does not provide authenticity!

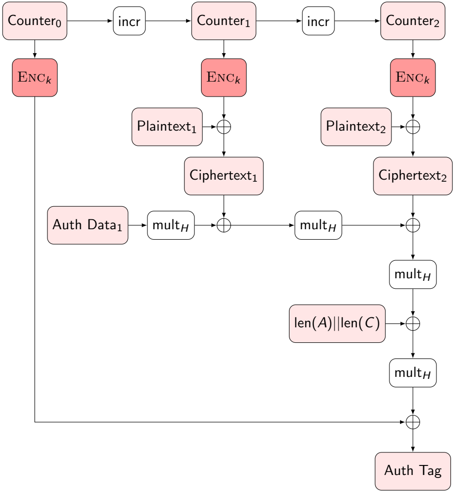
Real-world security

- Schemes providing authenticated encryption are IND-CCA secure.
- For details, see presentation linked from course Web site at

  `https://grothoff.org/christian/teaching/2024/4201/`

Part III: Real-world symmetric encryption

GCM encryption



Using encryption APIs

GNU libgcrypt is a C library offering a wide range of cryptographic primitives.

1. `# apt install libgcrypt20-dev`
2. `# apt install gcc gdb valgrind emacs`
3. Download source templates (`exercise.tgz`) from course Git

Example: AES256 GCM (encrypt.c)

```
char key[256/8], iv[96/8];
char plaintext[] = "Hello world";
char ciphertext[sizeof (plaintext)];
gcry_cipher_hd_t cipher;

gcry_cipher_open (&cipher, GCRY_CIPHER_AES256,
          GCRY_CIPHER_MODE_GCM, 0);
gcry_cipher_setkey (cipher, key, sizeof (key));
gcry_cipher_setiv  (cipher, iv,  sizeof (iv));
gcry_cipher_encrypt (cipher,
       ciphertext, sizeof (ciphertext),
       plaintext,  sizeof (plaintext));
gcry_cipher_close (cipher);
```

Example: AES256 GCM (decrypt.c)

```
char key[256/8], iv[96/8];
char plaintext[1024];
char ciphertext[sizeof (plaintext)];
gcry_cipher_hd_t cipher;

size_t plen = read (STDIN_FILENO,
                    ciphertext, sizeof (ciphertext));
gcry_cipher_open (&cipher, GCRY_CIPHER_AES256,
          GCRY_CIPHER_MODE_GCM, 0);
gcry_cipher_setkey (cipher, key, sizeof (key));
gcry_cipher_setiv  (cipher, iv,  sizeof (iv));
gcry_cipher_decrypt (cipher,
       plaintext,  plen,
       ciphertext, plen);
gcry_cipher_close (cipher);
```

Handling partial reads (decrypt.c)

```
char plaintext[1024];
size_t plen = 0;

while (1) {
  ssize_t inlen = read (STDIN_FILENO,
                        &ciphertext[plen],
                        sizeof (ciphertext) - plen);
  if (-1 == inlen) {
    fprintf (stderr,
             "Failed to read input\n");
    return 1;
  }
  if (0 == inlen)
    break;
  plen += inlen;
}
```

Tasks (1/3)

- Use the provided `encrypt` and `decrypt` programs to encrypt "Hello world" text using AES256+GCM and then decrypt it.
- Study the `libgcrypt` documentation. Use it to switch the program to use AES256+CBC instead.
- Switch back to AES256+GCM. Extend the program to obtain, transmit and verify the authentication tag.
- Extend the program to authenticate additional plaintext data that is not at all encrypted.

Tasks (2/3)

- Write a new program `hash.c` to compute the SHA-256 hash of the data read from `stdin`. Output the result in HEX and compare to `sha256sum`.
- Modify your program to use `SHA-512` instead.
- Write a new program `kdf.c` to compute the SCRYPT key derivation function. Output the result in HEX.

Tasks (3/3)

- Modify your programs to perform 10000 iterations each time before generating any output.
- Measure the time the various operations take.
- Modify your programs to process 1 MB of input instead of the 11 bytes of “Hello world”.
- Again, measure the time the various operations take.
- Change the IV length from 96 bits to 128 bits for AES256+GCM and measure again.

Break

Part IV: Symmetric key establishment protocols

Key Establishment Security goals

The basic security goals of key establishment are:

- *Key secrecy:* Session keys must not be known by anyone else than Alice, Bob (and maybe some trusted third party). Mallory must not learn anything about session keys.
- *Authenticity:* One party can be assured about the identity of the other party it shares the session key with. That is, Alice knows that she has session key with Bob.
- *Freshness of keys:* Mallory must not be able to replay old session keys.

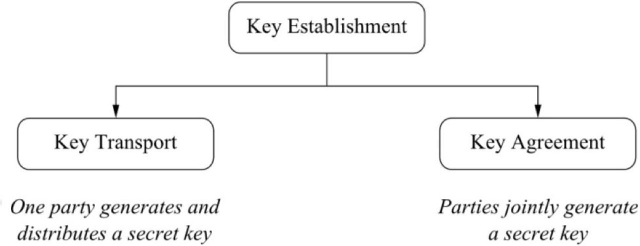
Protocols

- *Key establishment* is realized by using *protocols* whereby a shared secret becomes available to two or more parties, for subsequent cryptographic use.
- Until now, we have been discussing non-interactive crypto primitives, in the following we look at crypto protocols.
- It is *even harder* to design secure protocols, than designing non-interactive primitives. In fact, there is a long list of protocols designed by famous (and not so famous) cryptographers that were found to be flawed.

Session keys

- Key establishment protocols result *in shared secrets* which are typically called (or used to derive) session keys.
- Ideally, a session key is an ephemeral secret, i.e., one whose use is *restricted to a short time period such as a single telecommunications connection (or session)*, after which all trace of it is eliminated.
- Motivation for ephemeral keys includes the following:
  1. To limit available ciphertext (under a fixed key) for cryptanalytic attack;
  2. To limit exposure, with respect to both time period and quantity of data, in the event of (session) key compromise;
  3. To avoid long-term storage of a large number of distinct secret keys by creating keys only when actually required;
  4. To create independence across communications sessions or applications.

Classification of key establishment methods



Key Establishment

Key Transport

One party generates and distributes a secret key

Key Agreement

Parties jointly generate a secret key

Chapter 13 of Understanding Cryptography by Christof Paar and Jan Pelzl

Private channels

- Let us informally refer to a *private channel* as an authentic and confidential channel.
    - Exchange of secret keys on a USB stick
    - Pre-installation of keys on a company laptop
- Symmetric key distribution is impossible without private channels.
- Private channels are, loosely speaking, “complicated”, “inefficient”, “expensive”.
- The goal in the following is to:
    - *Reduce the number* of private channels required to exchange keys.
    - Use an *initial private channel* today to exchange a secret key that they may use *tomorrow for establishing a secure channel over an insecure link*.

Storytime

Once upon a time...

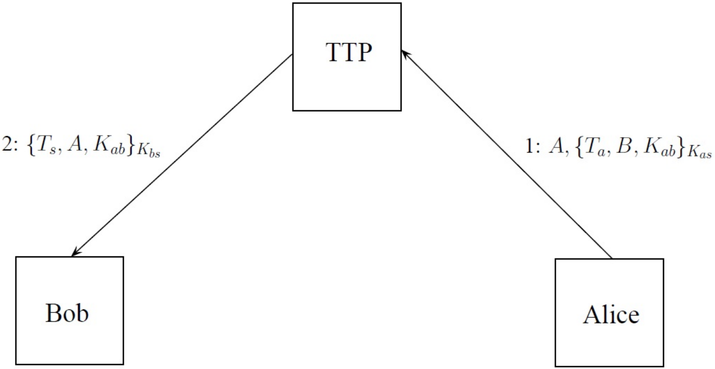
Neumann-Stubblebine

1. Alice sends A, R_A to Bob.
2. Bob sends $B, R_B, E_B(A, R_A, T_B)$ to Trent, where T_B is a timestamp and E_B uses a key Bob shares with Trent.
3. Trent generates random session key K and sends $E_A(B, R_A, K, T_B), E_B(A, K, T_B), R_B$ to Alice where E_A uses a key Alice shares with Trent.
4. Alice decrypts and confirms that R_A is her random value. She then sends to Bob $E_B(A, K, T_B), E_K(R_B)$.
5. Bob extracts K and confirms that T_B and R_B have the same value as in step 2.

Denning-Sacco

1. Alice sends A, B to Trent
2. Trent sends Alice $S_T(B, K_B), S_T(A, K_A)$
3. Alice sends Bob $E_B(S_A(K, T_A)), S_T(B, K_B), S_T(A, K_A)$
4. Bob decrypts, checks signatures and timestamps

Wide-Mouth Frog protocol

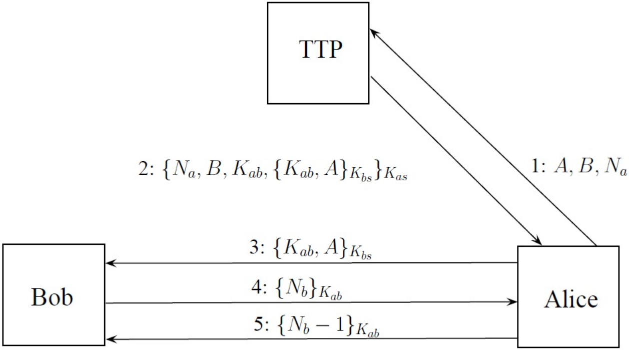

TTP
2: $\{T_s, A, K_{ab}\}_{K_{bs}}$
1: $A, \{T_a, B, K_{ab}\}_{K_{as}}$
Bob
Alice

Wide-Mouth Frog protocol

The wide-mouth frog protocol has some conceptual shortcomings:

- Assumes synchronized clocks between the parties to achieve freshness.
- Although having synchronized clocks seems to be straight-forward, this is actually not the case.
    - Synchronized clocks under normal conditions is indeed easy (you have that in Windows, Linux...).
    - Synchronized clocks under attack is much harder: you need to have another protocol that securely synchronizes clocks.
    - But as soon as clock synchronization becomes security relevant, you can bet that it gets attacked.
- Bob must trust Alice that she correctly generates the session key.

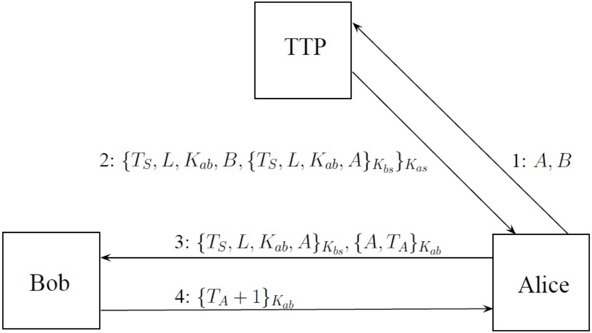
Needham-Schroeder protocol


TTP
1: A, B, N_a
2: $\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3: $\{K_{ab}, A\}_{K_{bs}}$
4: $\{N_b\}_{K_{ab}}$
5: $\{N_b - 1\}_{K_{ab}}$
Bob
Alice

Needham-Schroeder protocol

- Needham is one of the IT security pioneers. Protocol was conceived in 1978 and is one of the most widely studied security protocols ever.
- Removes timestamps and introduces nonces to achieve freshness.
- The session keys are generated by TTP in on the previous slide, thus removes problem of Wide-Mouth Frog protocol.
- Protocol is insecure against *known session key attacks*. Adversary who gets session key can replay the last three messages and impersonate A to B.
  - The reason for this problem is that B does not know whether the session key is fresh.
  - This vulnerability was discovered only some times after the protocol was published. Thus, even the smartest and most experienced people can fail to design secure crypto protocols.

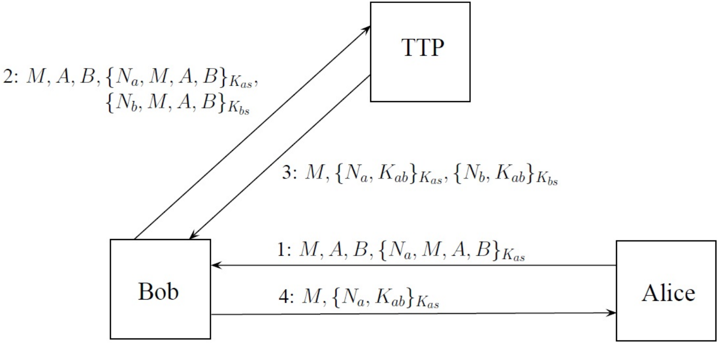
Kerberos


TTP
2: $\{T_S, L, K_{ab}, B, \{T_S, L, K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
1: A, B
3: $\{T_S, L, K_{ab}, A\}_{K_{bs}}, \{A, T_A\}_{K_{ab}}$
Bob
4: $\{T_A + 1\}_{K_{ab}}$
Alice

Kerberos

- Developed at MIT around 1987, made it into Windows 2000, and is still used as the authentication / key establishment / authorization mechanism within Windows.
- Quite similar to Needham-Schroeder, but removes weakness against known session key attacks using synchronized clocks.
- Shorter than Needham-Schroeder: only 4 messages instead of 5.

Otway-Rees protocol



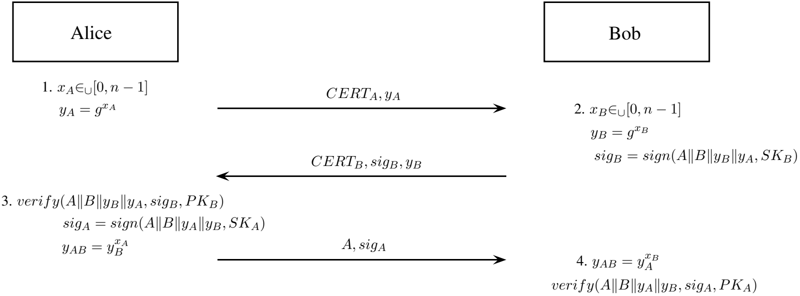
TTP
2: $M, A, B, \{N_a, M, A, B\}_{K_{as}},$
$\{N_b, M, A, B\}_{K_{bs}}$
3: $M, \{N_a, K_{ab}\}_{K_{as}}, \{N_b, K_{ab}\}_{K_{bs}}$
1: $M, A, B, \{N_a, M, A, B\}_{K_{as}}$
Bob
4: $M, \{N_a, K_{ab}\}_{K_{as}}$
Alice

Otway-Rees protocol

- Only 4 messages as Kerberos, but completely different messages.
- Does not require clock synchronization.
- Has a number of problems $\Rightarrow$ Homework!

Station to station key agreement protocol

Common input: $\mathbb{Z}^*{}_p$ and $g \in \mathbb{Z}^*{}_p$, and n such that $g^n \equiv 1 \mod p$



Alice

Bob

1. $x_A {\in_\cup} [0, n-1]$
$y_A = g^{x_A}$

$CERT_A, y_A$

2. $x_B {\in_\cup} [0, n-1]$
$y_B = g^{x_B}$
$sig_B = sign(A \| B \| y_B \| y_A, SK_B)$

$CERT_B, sig_B, y_B$

3. $verify(A \| B \| y_B \| y_A, sig_B, PK_B)$
$sig_A = sign(A \| B \| y_A \| y_B, SK_A)$
$y_{AB} = y_B^{x_A}$

A, sig_A

4. $y_{AB} = y_A^{x_B}$
$verify(A \| B \| y_A \| y_B, sig_A, PK_A)$

- The protocol above is a simplified version of the STS protocol to illustrate the idea of authenticating messages with public keys.
- For a detailed spec refer to `http://en.wikipedia.org/wiki/Station-to-Station_protocol`

Station to station key agreement protocol

- The “station to station protocol” is the DH protocol made secure against MIM attacks:
  - The idea is simple: Alice and Bob basically sign all the messages they exchange in the Diffie - Hellman protocol.
  - The “exchange of authenticated signing keys” is done using certificates.
- Station to station protocol is the basis for the practically important *IKE* (Internet Key Exchange protocol).
- The bottom line is: one cannot establish authenticated keys without bootstrapping the system using an “exterior authentication mechanism” (e.g., without first establishing public key certificates for Alice and Bob).

RSA key transport

`https://www.theinquirer.net/inquirer/news/2343117/ietf-drops-rsa-key-transport-from-ssl`

Lessons Learned

- Do not try to be too clever, over-optimization is often the cause for vulnerabilities
- Which optimizations you can do (and which optimization actually matter) depends on your assumptions (adversary model, system capabilities)
- Which protocol to use depends on your performance goals and communications capabilities (all-to-all communication, trusted party, latency, bandwidth and computational constraints)

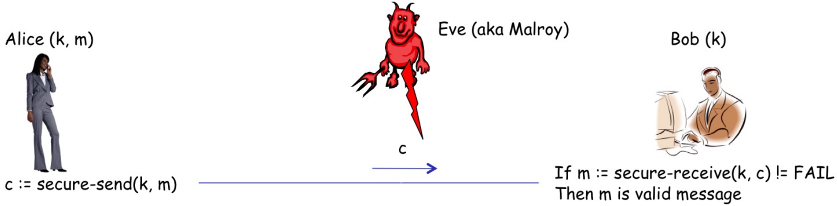
Break

Part V: Secure Channels

Overview

- By *secure channel* we refer to a logical channel running on top of some insecure link (typically the Internet) that provides
  - Confidentiality
  - Integrity and authenticity
  - Message freshness
- Secure channels are probably one of the most important applications of crypto in the real world.
- Many well known secure network protocols such as TLS/SSL, VPNs, IPSec, WPA etc but also application specific (e.g., secure VoIP), and proprietary protocols (maybe Skype?) make use of secure channels.
- Essentially all these protocols build upon the basic ideas we discuss in the following.
- It is also possible to get it wrong, e.g., the WEP protocol has a series of security flaws.

Secure channel



Alice (k, m)

c := secure-send(k, m)

Eve (aka Malroy)

c

Bob (k)

If m := secure-receive(k, c) != FAIL
Then m is valid message

Secure channel - Secure send

```
secure-send($m$, $k_E$, $k_M$) {
    STATIC $msgsnt := 1$
    IF ($msgsnt \geq MAX_{MSGS}$) THEN RETURN $\perp$
    $c := ENC(k_E, m)$
    $\tilde{m} := msgsnt||LENGTH(c)||c$
    $t := MAC(k_M, \tilde{m})$
    $SEND(\tilde{m}||t)$
    $msgsnt := msgsnt + 1$
}
```

Secure channel - Secure receive

```
secure-receive($C$, $k_E$, $k_M$) {
    STATIC $msgrcvd := 0$
    $(msgsnt, len, c, t) = PARSE(C)$
    IF ($t \neq MAC(k_M, msgsnt||len||c)$) THEN RETURN $\perp$
    IF ($msgsnt \leqslant msgrcvd$) THEN RETURN $\perp$
    $m := DEC(k_E, c)$
    $msgrcvd := msgsnt$
    RETURN m
}
```

Remarks

- The *freshness property* based on counters guarantees the following: If $m_1, m_2, \ldots, m_n$ denote the messages send using secure-send(), then secure-receive() can guarantee that the messages $m_1, m_2, \ldots, m_n$ being received are subsequence of the messages sent.
- Counters give no timing guarantees, i.e., the adversary Mallory can delay messages at will.
- Timing guarantees can be achieved using
  - Time-stamps
  - Challenges
- No security protocol can prevent Mallory from discarding messages.
- MACs provide not just integrity protection but also *authenticity*, as discussed earlier.
- Further reading material: Chapter 8 in Practical Cryptography by Schneier & Ferguson.

Remarks

- Typically, secure-send() and secure-receive() are run by both parties using a secure channel.
- Each party will have an independent key-pair (enc & MAC).
- In practice, one introduces the notion of a session (e.g., e-banking). Consists of a session ID in the header, which allows the receiver to look-up session state (keys, counters etc.) when receiving a message.
- Generally better is the use of authenticated encryption, where the block-cipher mode guarantees confidentiality **and** integrity.
- For more info see last week's slides on AES-GCM and `http://en.wikipedia.org/wiki/Authenticated_encryption`

Break

Part IV: Extended Security Objectives for Secure Channels

Repudiation vs. non-repudiation

- Digital signatures allow *proving* that someone said something
- Alice may be happy to authenticate to Bob, but not to Eve or Mallory!

Repudiation vs. non-repudiation

- Digital signatures allow *proving* that someone said something
- Alice may be happy to authenticate to Bob, but not to Eve or Mallory!
- Bob may turn "evil" and use Alice's statements against her later

⇒ Signatures may provide too much (authentication *and* non-repudiation)

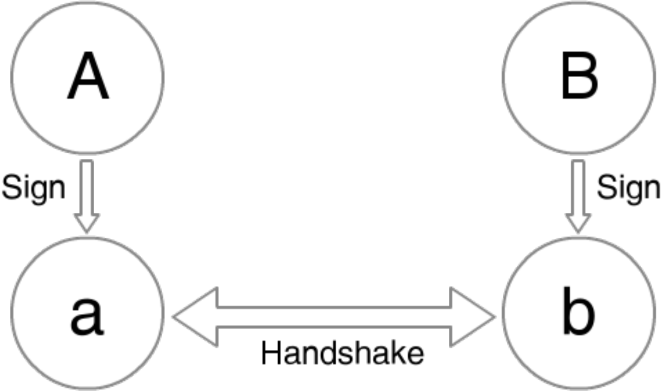
Off-the-record (OTR) protocols allow *repudiation*

OTR (Idea)

$$S_A(T_A) \tag{1}$$

$$S_B(T_B) \tag{2}$$

$$HKDF(DH(T_A, T_B)) \tag{3}$$



OTR (Real)

The OTR protocol protects the above KX by wrapping it inside another ephemeral key exchange:

$$K_1 := DH(T_A^1 || T_B^1) \tag{4}$$

$$E_{K_1}(S_A(T_A^2)) \tag{5}$$

$$E_{K_1}(S_B(T_B^2)) \tag{6}$$

$$K_2 := HKDF(DH(T_A^2, T_B^2)) \tag{7}$$

$$\tag{8}$$

To achieve forward secrecy, OTR keeps rolling out new keys $T_{A,B}^i$. To improve deniability, OTR publishes the old MAC keys once the conversation progresses.

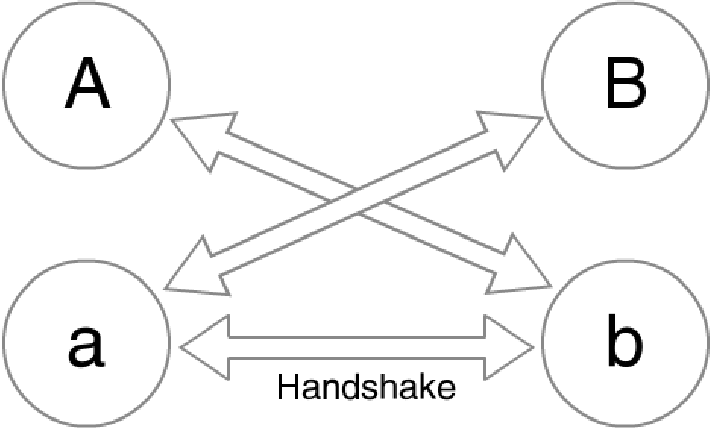
Is OTR deniable?

Is OTR deniable?

Both parties still have proof that they communicated: $S_X(T_X)$!

3DH (Trevor Perrin)

A: $K = HKDF(DH(T_a, T_B)||DH(T_a, B)||DH(a, T_B))$
B: $K = HKDF(DH(T_A, T_b)||DH(T_A, b)||DH(A, T_b))$



A Message from God (Dominic Tarr)

With 3DH, what happens if Alice's private key (a, T_a) is compromised?

A Message from God (Dominic Tarr)

With 3DH, what happens if Alice's private key (a, T_a) is compromised?

$$\text{M: } K = HKDF(DH(T_a, T_G)||DH(T_a, G)||DH(a, T_G))$$

$$\text{A: } K = HKDF(DH(T_a, T_G)||DH(T_a, G)||DH(a, T_G))$$

Forward secrecy

What happens if your private key is compromised
to your *past* communication data?

Static keys vs. ephemeral keys

Diffie-Hellman with:

- static keys allow authenticated encryption without signatures
- ephemeral keys protect against replay attacks and provide forward secrecy

Part VI: Full Spectrum Cyber

Hardware

General notions:

- Platforms with disabled Intel ME & disabled remote administration are safer.
- Platforms using uncommon CPU archtectures (Power7, Sparc) are safer.
- VMs are not a security mechanism. Side-channel attacks abound. Avoid running any software in a virtual machine “for security”.

Operating system

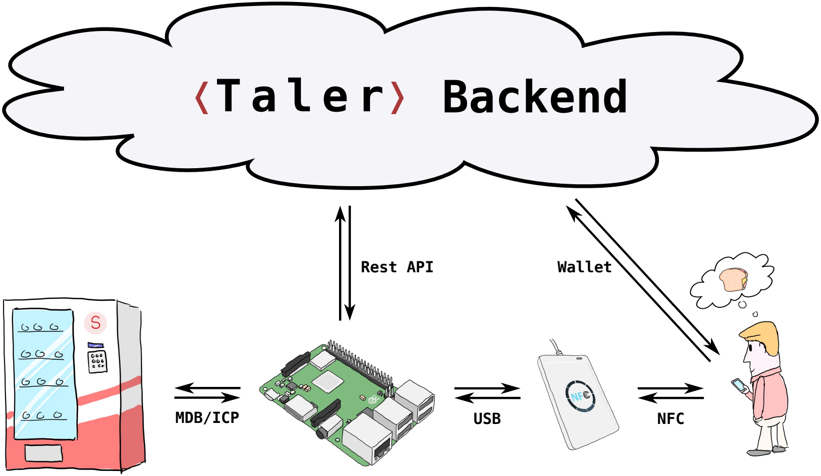
General notions:

- It should be safe to run different reasonably secure components (such as Nginx and Postgres) on the same physical hardware (under different UIDs/GIDs). You may want to separate them onto different physical machines during scale-out, but not necessarily for "basic" security.
- Limiting and auditing system administrator access will be crucial.
- Recommend to **not** use any anti-virus: more of a liability than an asset.
- Recommend using a well-supported GNU/Linux operating system (such as Debian or Ubuntu or Nix).

Part VII: Outlook

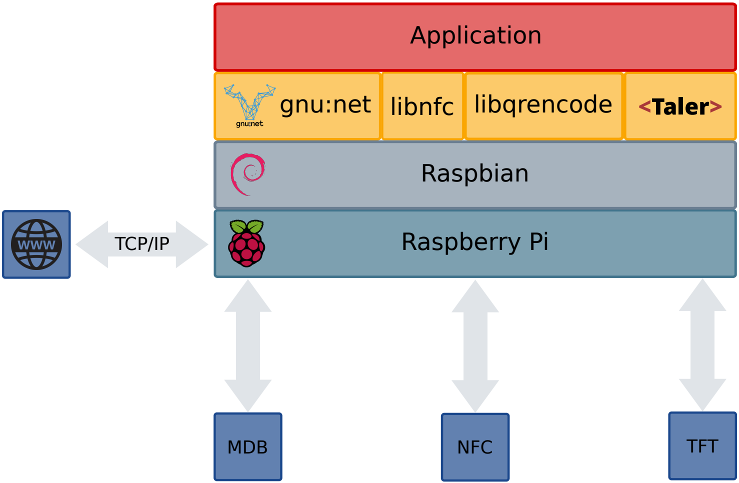
The Taler Snack Machine

Integration of a MDB/ICP to Taler gateway.
Implementation of a NFC or QR-Code to Taler wallet interface.

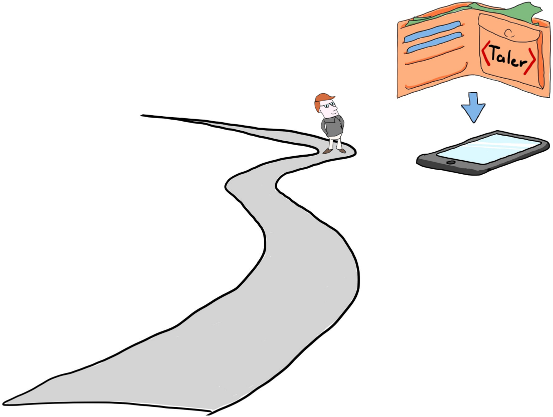


by M. Boss and D. Hofer

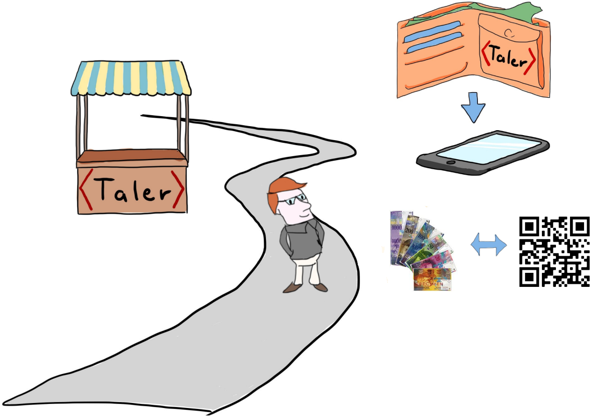
Software architecture for the Taler Snack Machine



Exercise: Install App on Smartphone



Exercise: Withdraw e-cash



Exercise: Use machine!