6.1800 Spring 2024

Lecture #23: Secure Channels

confidentiality and integrity through the magic of cryptography
so far, we’ve dealt with adversaries that were trying to access data on a server
so far, we’ve dealt with adversaries that were trying to access data on a server
this week, we’re going to turn to adversaries that are observing data on the network.
this week, we’re going to turn to adversaries that are observing data on the network

some network traffic is difficult to interpret
e.g., IP addresses are private or resolve to Akamai or Amazon servers
this week, we’re going to turn to adversaries that are observing data on the network

some network traffic is difficult to interpret
e.g., IP addresses are private or resolve to Akamai or Amazon servers

14:05:31.983557 34392425us tsft -62dB signal -98dB noise antenna 1 5785 MHz 11a ht/20 [bit 20] CF +QoS IP 184.28.89.95.443 > 10.189.86.146.41204: Flags [P.], seq 1643649202:1643649233, ack 1215791031, win 285, options [nop,nop,TS val 2235675295 ecr 95087166], length 31

0x0000: aaaa 0300 0000 0800 4548 0053 b11e 0000 ........EH..@.
0x0010: 3506 2174 b81c 595f 0abd 5692 01bb a0f4 5.!t..Y_.V.....
0x0020: 61f8 18b2 4877 7fb7 8018 011d 835f 0000 a...Hw.........
0x0030: 0101 080a 8541 b29f 05aa ea3e 1503 0300 .....A....>.....
0x0040: 1ac6 d28d 46ab 64f6 36a3 4efb edd1 f693 .....F.d.6.N.....
0x0050: 5cf0 0132 65f2 0b0d 21dd 66 \..e.../.f
this week, we're going to turn to adversaries that are observing data on the **network**

**some network traffic is difficult to interpret**
e.g., IP addresses are private or resolve to Akamai or Amazon servers

---

14:05:31.983557 34392425us tsft -62dB signal -98dB noise antenna 1 5785 MHz 11a
ht/20 [bit 20] CF +QoS IP 184.28.89.95.443 > 10.189.86.146.41204: Flags [P.], seq 1643649202:1643649233, ack 1215791031, win 285, options [nop,nop,TS val 2235675295 ecr 95087166], length 31

0x0000: aaaa 0300 0000 0000 4548 0053 b11e 4000 ........EH.S..@
0x0010: 3506 2174 b81c 595f 0abd 5692 01bb 0a0f 5.!t..Y_.V.....
0x0020: 61f8 18b2 4877 7fb7 8018 011d 835f 0000 a...Hw........_
0x0030: 0101 080a 8541 b29f 05a5 ea3e 1503 0300 ....A...>.-
0x0040: 1ac6 d28d 46ab 64f6 36a3 4efb edd1 f693 ....F.d.6.N......
0x0050: 5cf0 0132 65f2 0b0d 21dd 66 \..2e..!f

[katrina ~] dig -x 184.28.89.95
; <>> DIG 9.8.3-P1 <>> -x 184.28.89.95
; global options: +cmd
; Got answer:
; ->HEADER<< opcode: QUERY, status: NOERROR, id: 47850
; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 8, ADDITIONAL: 8

; QUESTION SECTION:
95.89.28.184.in-addr.arpa. IN PTR

; ANSWER SECTION:
95.89.28.184.in-addr.arpa. 43125 IN PTR a184-28-89-95.deploy.static.akamaitechnologies.com.
this week, we’re going to turn to adversaries that are observing data on the network. Some packet data can reveal what you’re doing even if the packet headers are difficult to interpret.
this week, we’re going to turn to adversaries that are observing data on the network

some packet data can reveal what you’re doing even if the packet headers are difficult to interpret
this week, we’re going to turn to adversaries that are observing data on the network

some packet data can reveal what you’re doing even if the packet headers are difficult to interpret
this week, we’re going to turn to adversaries that are observing data on the network.

Some packet data can reveal what you’re doing even if the packet headers are difficult to interpret.
this week, we’re going to turn to adversaries that are observing data on the network

sometimes traffic can be easily tied to individuals
either in packet headers or packet data
this week, we’re going to turn to adversaries that are observing data on the network sometimes traffic can be easily tied to individuals either in packet headers or packet data

14:05:29.947459 104653458us tsft -70dB signal -92dB noise antenna 0 2412 MHz 11g ht/20 39.0 Mb/s MCS 10 20 MHz lon GI mixed BCC FEC [bit 20] CF +QoS IP 10.189.6.135.5353 > 224.0.0.251.5353: 0*- [0q] 2/0/3 (Cache flush) PTR Bobs-iPhone.local., (Cache flush) PTR Bobs-iPhone.local. (217)

```
0x0000:  aaaa 0300 0000 0080 4500 00f5 2053 0000 ........E....S..
0x0010:  ff11 a865 0abd 0687 e000 00fb 14e9 14e9 ...e........
0x0020:  00e1 5867 0000 8400 0000 0002 0000 0003 ..Xg........
0x0030:  0137 0135 0144 0133 0139 0130 0138 0133 .7.5.D.3.9.0.8.3
0x0040:  0135 0135 0139 0144 0144 0141 0143 0130 .5.5.9.D.D.A.C.0
0x0050:  0130 0130 0130 0130 0130 0130 0130 0130 .0.0.0.0.0.0.0.0
0x0060:  0130 0130 0130 0130 0130 0130 0138 0145 0146 .0.0.0.0.8.E.F
0x0070:  0369 7036 0461 7270 6100 000c 8001 0000 .ip6.arpa.......x.....Bobs-iPho
0x0080:  0078 0002 0000 0078 0002 0000 0002 0000 ..x.........u
0x0090:  0000 2905 a000 0011 9400 1200 0400 0e00 ................%n..}..1}..
this week, we’re going to turn to adversaries that are observing data on the network

sometimes traffic can be easily tied to individuals either in packet headers or packet data
this week, we’re going to turn to adversaries that are observing data on the network

sometimes traffic can be easily tied to individuals either in packet headers or packet data

today we’re going to focus on how to protect packet data from an adversary next time, we’ll talk about how you can protect meta-information (e.g., packet headers) from an adversary
policy: provide confidentiality (adversary cannot learn message contents) and integrity (adversary cannot tamper with packets and go undetected)
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

encrypt\((key, \text{ message})\) → ciphertext

decrypt\((key, ciphertext)\) → message

encrypt\(34fbcbd1, \text{“hello, world”}\) = 0x47348f63a67926cd393d4b93c58f78c

decrypt\(34fbcbd1, \text{“0x47348f63a67926cd393d4b93c58f78c”}\) = hello, world
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[
\text{encrypt(key, message)} \rightarrow \text{ciphertext} \\
\text{decrypt(key, ciphertext)} \rightarrow \text{message}
\]

\[
\text{encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c} \\
\text{decrypt(34fcbbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world}
\]

**property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

encrypt(key, message) → ciphertext

decrypt(key, ciphertext) → message

encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world

**property:** given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

encrypt(key, message) → ciphertext

decrypt(key, ciphertext) → message

encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world

**property:** given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key

adversary can't determine message, but might be able to cleverly alter ciphertext so that it decrypts to a different message
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

encrypt(key, message) \(\rightarrow\) ciphertext

decrypt(key, ciphertext) \(\rightarrow\) message

encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c") = hello, world

**property:** given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

encrypt(key, message) → ciphertext

decrypt(key, ciphertext) → message

encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world

**property:** given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key

no good — if the adversary changes ciphertext, it can also (correctly) update the hash
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[
\text{encrypt} (\text{key}, \text{message}) \rightarrow \text{ciphertext} \\
\text{decrypt} (\text{key}, \text{ciphertext}) \rightarrow \text{message}
\]

\[
\text{encrypt}(34fbcbd1, \text{"hello, world"}) = 0x47348f63a67926cd393d4b93c58f78c \\
\text{decrypt}(34fbcbd1, "0x47348f63a67926cd393d4b93c58f78c") = \text{hello, world}
\]

**property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

\[
\text{MAC} (\text{key}, \text{message}) \rightarrow \text{token}
\]

no good — if the adversary changes **ciphertext**, it can also (correctly) update the hash
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[
\text{encrypt}(\text{key}, \text{message}) \rightarrow \text{ciphertext} \\
\text{decrypt}(\text{key}, \text{ciphertext}) \rightarrow \text{message}
\]

\[
\text{encrypt}(34fbcbd1, \text{“hello, world”}) = 0x47348f63a67926cd393d4b93c58f78c \\
\text{decrypt}(34fbcbd1, \text{“0x47348f63a67926cd393d4b93c58f78c”}) = \text{hello, world}
\]

**property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

\[
\text{MAC}(\text{key}, \text{message}) \rightarrow \text{token}
\]

\[
\text{MAC}(34fbcbd1, \text{“hello, world”}) = 0x59cccc95723737f777e62bc756c8da5c
\]

no good — if the adversary changes **ciphertext**, it can also (correctly) update the hash
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[
\text{encrypt}(\text{key}, \text{message}) \rightarrow \text{ciphertext} \\
\text{decrypt}(\text{key}, \text{ciphertext}) \rightarrow \text{message}
\]

encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world

**property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

\[
\text{MAC}(\text{key}, \text{message}) \rightarrow \text{token} \\
\text{MAC}(34fbcbd1, “hello, world”) = 0x59cccc95723737f777e62bc756c8da5c
\]

**property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

it is also impossible to go in the reverse direction: given **token**, you can’t get **message** even with the **key**

no good — if the adversary changes **ciphertext**, it can also (correctly) update the hash
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

### Encrypt/Decrypt

**encrypt** \( (key, \text{message}) \rightarrow \text{ciphertext} \)

**decrypt** \( (key, \text{ciphertext}) \rightarrow \text{message} \)

- \( \text{encrypt}(34fbcbd1, \text{“hello, world”}) = 0x47348f63a67926cd393d4b93c58f78c \)
- \( \text{decrypt}(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = \text{hello, world} \)

**property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

### MAC

**MAC** \( (key, \text{message}) \rightarrow \text{token} \)

- \( \text{MAC}(34fbcbd1, \text{“hello, world”}) = 0x59cccc95723737f777e62bc756c8da5c \)

**property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

It is also impossible to go in the reverse direction: given **token**, you can’t get **message** even with the **key**
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

<table>
<thead>
<tr>
<th>encrypt(key, message)</th>
<th>ciphertext</th>
</tr>
</thead>
<tbody>
<tr>
<td>decrypt(key, ciphertext)</td>
<td>message</td>
</tr>
</tbody>
</table>

\[
\text{encrypt}(34fbcbd1, \text{"hello, world"}) = 0x47348f63a67926cd393d4b93c58f78c
\]
\[
\text{decrypt}(34fbcbd1, "0x47348f63a67926cd393d4b93c58f78c") = \text{hello, world}
\]

**property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

<table>
<thead>
<tr>
<th>alice</th>
</tr>
</thead>
<tbody>
<tr>
<td>c = encrypt(k, m)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>h = MAC(k, c)</td>
</tr>
</tbody>
</table>

**MAC(key, message) → token**
\[
\text{MAC}(34fbcbd1, \text{"hello, world"}) = 0x59cccc95723737f777e62bc756c8da5c
\]

**property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

it is also impossible to go in the reverse direction: given **token**, you can’t get **message** even with the **key**
Threat model: adversary can observe network data, tamper with packets, and insert its own packets

**Policy:** provide confidentiality (adversary cannot learn message contents) and integrity (adversary cannot tamper with packets and go undetected)

Encrypt(key, message) → ciphertext

\[
\text{encrypt}(34fbcbd1, \text{"hello, world"}) = 0x47348f63a67926cd393d4b93c58f78c
\]

Decrypt(key, ciphertext) → message

\[
\text{decrypt}(34fbcbd1, \text{"0x47348f63a67926cd393d4b93c58f78c"}) = \text{hello, world}
\]

**Property:** given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key

\[
\text{MAC}(34fbcbd1, \text{"hello, world"}) = 0x59cccc95723737f777e62bc756c8da5c
\]

Alice

\[
\begin{align*}
c &= \text{encrypt}(k, m) \\
h &= \text{MAC}(k, c)
\end{align*}
\]

Bob

In practice, we’d use one key to encrypt and a different one to MAC

Message:

Encrypt(key, message) → ciphertext

\[
\text{encrypt}(34fbcbd1, \text{"hello, world"}) = 0x47348f63a67926cd393d4b93c58f78c
\]

Decrypt(key, ciphertext) → message

\[
\text{decrypt}(34fbcbd1, \text{"0x47348f63a67926cd393d4b93c58f78c"}) = \text{hello, world}
\]

Property:

Given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key. It is also impossible to go in the reverse direction: given token, you can’t get message even with the key.
policy: provide confidentiality (adversary cannot learn message contents) and integrity (adversary cannot tamper with packets and go undetected)

threat model: adversary can observe network data, tamper with packets, and insert its own packets

encrypt(key, message) → ciphertext
decrypt(key, ciphertext) → message

encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world

property: given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key

MAC(key, message) → token
MAC(34fbcbd1, “hello, world”) = 0x59cccc95723737f777e62bc756c8da5c

property: given the message, it is (virtually) impossible to obtain the token without knowing the key

it is also impossible to go in the reverse direction: given token, you can’t get message even with the key

in practice, we’d use one key to encrypt and a different one to MAC

Alice

\[ c = \text{encrypt}(k, m) \]

\[ h = \text{MAC}(k, c) \]

Bob

\[ c | h \]
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[ \text{encrypt}(\text{key}, \text{message}) \rightarrow \text{ciphertext} \]
\[ \text{decrypt}(\text{key}, \text{ciphertext}) \rightarrow \text{message} \]

\[ \text{encrypt}(34\text{fbcbd1}, \text{"hello, world"}) = 0x47348f63a67926cd393d4b93c58f78c \]
\[ \text{decrypt}(34\text{fbcbd1}, \text{"0x47348f63a67926cd393d4b93c58f78c"}) = \text{hello, world} \]

**property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

\[ \text{MAC}(\text{key}, \text{message}) \rightarrow \text{token} \]
\[ \text{MAC}(34\text{fbcbd1}, \text{"hello, world"}) = 0x59cccc95723737f777e62bc756c8da5c \]

**property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

It is also impossible to go in the reverse direction: given **token**, you can’t get **message** even with the **key**
**Policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**Threat Model:** adversary can observe network data, tamper with packets, and insert its own packets

$$\text{encrypt(}k, \text{ message}) \rightarrow \text{ciphertext}$$
$$\text{decrypt(}k, \text{ ciphertext}) \rightarrow \text{message}$$

$\text{encrypt(34fbcbd1, "hello, world") = 0x47348f63a67926cd393d4b93c58f78c}$
$\text{decrypt(34fbcbd1, "0x47348f63a67926cd393d4b93c58f7 \8c") = hello, world}$

**Property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

$$\text{MAC(}k, \text{ message}) \rightarrow \text{token}$$
$$\text{MAC(34fbcbd1, "hello, world") = 0x59cccc95723737f777e62bc756c8da5c}$$

**Property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

It is also impossible to go in the reverse direction: given **token**, you can't get **message** even with the **key**
**Policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**Threat Model:** adversary can observe network data, tamper with packets, and insert its own packets

\[
\text{encrypt}(\text{key}, \text{message}) \rightarrow \text{ciphertext}
\]

\[
\text{decrypt}(\text{key}, \text{ciphertext}) \rightarrow \text{message}
\]

\[
\text{encrypt}(34fbcbd1, \text{"hello, world"}) = 0x47348f63a67926cd393d4b93c58f78c
\]

\[
\text{decrypt}(34fbcbd1, \text{"0x47348f63a67926cd393d4b93c58f78c"}) = \text{hello, world}
\]

**Property:** given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key.

\[
\text{MAC}(\text{key}, \text{message}) \rightarrow \text{token}
\]

\[
\text{MAC}(34fbcbd1, \text{"hello, world"}) = 0x59cccc95723737f777e62bc756c8da5c
\]

**Property:** given the message, it is (virtually) impossible to obtain the token without knowing the key.

It is also impossible to go in the reverse direction: given token, you can’t get message even with the key.

**Eve:** can neither read \(m\) nor tamper with \(c\) (without going unnoticed)
**Policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**Threat Model:** adversary can observe network data, tamper with packets, and insert its own packets

encrypt(key, message) → ciphertext

decrypt(key, ciphertext) → message

\[
c = \text{encrypt}(k, m)
\]
\[
h = \text{MAC}(k, c)
\]
\[
\text{MAC}(k, c) == h ?
\]
\[
m = \text{decrypt}(k, c)
\]

**Property:** given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key

\[
\text{MAC}(\text{key}, \text{message}) \rightarrow \text{token}
\]
\[
\text{MAC}(34fbcbd1, \text{"hello, world"}) = 0x59cccc95723737f777e62bc756c8da5c
\]

**Property:** given the message, it is (virtually) impossible to obtain the token without knowing the key

\[
\text{it is also impossible to go in the reverse direction: given token, you can’t get message even with the key}
\]
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[
\begin{align*}
\text{encrypt}(\text{key}, \text{message}) & \rightarrow \text{ciphertext} \\
\text{decrypt}(\text{key}, \text{ciphertext}) & \rightarrow \text{message}
\end{align*}
\]

\[
\begin{align*}
\text{encrypt}(34fbcbd1, \text{“hello, world”}) = 0x47348f63a67926cd393d4b93c58f78c \\
\text{decrypt}(34fbcbd1, \text{“0x47348f63a67926cd393d4b93c58f78c”}) = \text{hello, world}
\end{align*}
\]

**property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

\[
\begin{align*}
\text{c} = \text{encrypt}(k, m) \\
\text{h} = \text{MAC}(k, c)
\end{align*}
\]

\[
\begin{align*}
\text{MAC}(k, c) \Rightarrow \text{h} \\
\text{m} = \text{decrypt}(k, c)
\end{align*}
\]

**property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

\[
\begin{align*}
\text{MAC}(\text{key}, \text{message}) & \rightarrow \text{token} \\
\text{MAC}(34fbcbd1, \text{“hello, world”}) = 0x59cccc95723737f777e62bc756c8da5c
\end{align*}
\]

**property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

it is also impossible to go in the reverse direction: given **token**, you can’t get **message** even with the **key**
**Policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**Threat Model:** adversary can observe network data, tamper with packets, and insert its own packets

```plaintext
encrypt(key, message) → ciphertext
decrypt(key, ciphertext) → message
```

```plaintext
encrypt(34fbcbd1, "hello, world") = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, "0x47348f63a67926cd393d4b93c58f78c") = hello, world
```

**Property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

```plaintext
MAC(key, message) → token
MAC(34fbcbd1, "hello, world") = 0x59cccc95723737f777e62bc756c8da5c
```

**Property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

It is also impossible to go in the reverse direction: given **token**, you can’t get **message** even with the **key**

**Problem:** **replay attacks**

*Eve* could intercept a message, re-send it at a later time
threat model: adversary can observe network data, tamper with packets, and insert its own packets

policy: provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**question:** why would **eve** do this? can you think of times when re-sending a message would cause damage? bonus question: do you know any techniques to mitigate this attack?

```
c = encrypt(k, m)

h = MAC(k, c)
```

```
MAC(k, c) == h ?
m = decrypt(k, c)
```

**property:** given the **cipher text**, it is (virtually) impossible to obtain the **message** without knowing the **key**

```
MAC(key, message) -> token
```

```
MAC(34fbcbd1, “hello, world”) = 0x59cccc95723737f777e62bc756c8da5c
```

**property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

it is also impossible to go in the reverse direction: given **token**, you can’t get **message** even with the **key**

```
encrypt(key, message) -> ciphertext
```

```
decrypt(key, ciphertext) -> message
```

```
encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world
```

```
c | h
```

```
MAC(34fbcbd1, “hello, world”) = 0x59cccc95723737f777e62bc756c8da5c
```

```
c | h
```

```
c = encrypt(k, m)
```

```
h = MAC(k, c)
```

```
c = decrypt(k, c)
```

```
h = MAC(k, c)
```
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[
\begin{align*}
\text{encrypt}(\text{key}, \text{message}) & \rightarrow \text{ciphertext} \\
\text{decrypt}(\text{key}, \text{ciphertext}) & \rightarrow \text{message}
\end{align*}
\]

\[
\begin{align*}
\text{c} = \text{encrypt}(k, m | \text{seq}) \\
h = \text{MAC}(k, c)
\end{align*}
\]

\[
\begin{align*}
\text{MAC}(k, c) & = h ? \\
m | \text{seq} & = \text{decrypt}(k, c)
\end{align*}
\]

**property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

\[
\begin{align*}
\text{MAC}(\text{key}, \text{message}) & \rightarrow \text{token} \\
\text{MAC}(34\text{fbcbd1}, \text{“hello, world“}) & = 0x59cccc95723737f777e62bc756c8da5c
\end{align*}
\]

**property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

it is also impossible to go in the reverse direction: given **token**, you can’t get **message** even with the **key**
**Policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**Threat Model:** adversary can observe network data, tamper with packets, and insert its own packets

\[ \text{encrypt}(\text{key}, \text{message}) \rightarrow \text{ciphertext} \]
\[ \text{decrypt}(\text{key}, \text{ciphertext}) \rightarrow \text{message} \]

- $c = \text{encrypt}(k, m \mid \text{seq})$
- $h = \text{MAC}(k, c)$
- $\text{MAC}(k, c) == h \ ?$
- $m \mid \text{seq} = \text{decrypt}(k, c)$

**Property:** given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key

**MAC:**
\[ \text{MAC}(\text{key}, \text{message}) \rightarrow \text{token} \]
- $\text{MAC}(34fbcbd1, \text{“hello, world”}) = 0x59cccc95723737f777e62bc756c8da5c$

**Property:** given the message, it is (virtually) impossible to obtain the token without knowing the key

It is also impossible to go in the reverse direction: given token, you can’t get message even with the key
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity**
(adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[
\text{encrypt}(\text{key}, \text{message}) \rightarrow \text{ciphertext} \\
\text{decrypt}(\text{key}, \text{ciphertext}) \rightarrow \text{message}
\]

\[
\begin{align*}
\text{c} &= \text{encrypt}(k, m \mid \text{seq}) \\
\text{h} &= \text{MAC}(k, \text{c})
\end{align*}
\]

property: given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

\[
\text{MAC}(\text{key}, \text{message}) \rightarrow \text{token}
\]

\[
\text{MAC}(34fbcbd1, \text{“hello, world”}) = \text{0x59cccc95723737f777e62bc756c8da5c}
\]

property: given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

it is also impossible to go in the reverse direction: given **token**, you can’t get **message** even with the **key**

if **eve** replays the message, **bob** will notice because **bob** has already seen this sequence number

\[
\begin{align*}
\text{c} \mid \text{seq} &= \text{decrypt}(k, \text{c}) \\
\text{MAC}(k, \text{c}) &= \text{h}?
\end{align*}
\]
**policy:** provide confidentiality (adversary cannot learn message contents) and integrity (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

encrypt(key, message) → ciphertext
decrypt(key, ciphertext) → message

\[
e \leftarrow c = \text{encrypt}(k, m | \text{seq})
\]

\[
h = \text{MAC}(k, c)
\]

\[
c | h \quad \text{MAC}(k, c) = h? \\
\]

\[
m | \text{seq} = \text{decrypt}(k, c)
\]

**property:** given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key

\[
\text{MAC}(key, message) \rightarrow \text{token}
\]

\[
\text{MAC}(34fbcbd1, \text{“hello, world”}) = \text{0x59cccc95723737f777e62bc756c8da5c}
\]

**property:** given the message, it is (virtually) impossible to obtain the token without knowing the key

it is also impossible to go in the reverse direction: given token, you can’t get message even with the key
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[
\text{encrypt}(\text{key, message}) \rightarrow \text{ciphertext} \\
\text{decrypt}(\text{key, ciphertext}) \rightarrow \text{message}
\]

\[
\text{encrypt}(34fbcbd1, \text{“hello, world”}) = \text{0x47348f63a67926cd393d4b93c58f78c} \\
\text{decrypt}(34fbcbd1, \text{“0x47348f63a67926cd393d4b93c58f78c”}) = \text{hello, world}
\]

**property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

\[
\text{MAC}(\text{key, message}) \rightarrow \text{token} \\
\text{MAC}(34fbcbd1, \text{“hello, world”}) = \text{0x59cccc95723737f777e62bc756c8da5c}
\]

**property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

It is also impossible to go in the reverse direction: given **token**, you can’t get **message** even with the **key**

**problem:** reflection attacks

eve could intercept a message, re-send it at a later time in the opposite direction
policy: provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

threat model: adversary can observe network data, tamper with packets, and insert its own packets

**encrypt(key, message) → ciphertext**

**decrypt(key, ciphertext) → message**

```plaintext
encrypt(34fbcbcd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbcd1, “0x47348f63a67926cd393d4b93c58f78c“) = hello, world
```

**property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

**MAC(key, message) → token**

```plaintext
MAC(34fbcbcd1, “hello, world”) = 0x59cccc95723737f777e62bc756c8da5c
```

**property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

it is also impossible to go in the reverse direction: given **token**, you can’t get **message** even with the **key**

<table>
<thead>
<tr>
<th>alice</th>
<th>bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_a = encrypt(k_a, m_a</td>
<td>seq_a)$</td>
</tr>
<tr>
<td>$h_a = MAC(k_a, c_a)$</td>
<td></td>
</tr>
<tr>
<td>$c_a</td>
<td>h_a$</td>
</tr>
<tr>
<td>$MAC(k_a, c_a) == h_a \ ?$</td>
<td></td>
</tr>
<tr>
<td>$m_a</td>
<td>seq_a = decrypt(k_a, c_a)$</td>
</tr>
</tbody>
</table>
**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

```
encrypt(key, message) → ciphertext
decrypt(key, ciphertext) → message
```

encrypt(34fbcbd1, “hello, world”) = 0x47348f63a679 26cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world

**property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

```
MAC(key, message) → token
MAC(34fbcbd1, “hello, world”) = 0x59cccc95723737f777e62bc756c8da5c
```

**property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

it is also impossible to go in the reverse direction: given **token**, you can’t get **message** even with the **key**
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[

e_{\text{alice}} = \text{encrypt}(k_a, m_a | \text{seq}_a) \\
h_a = \text{MAC}(k_a, c_a)
\]

\[
\text{MAC}(k_a, c_a) = h_a \? \\
m_a | \text{seq}_a = \text{decrypt}(k_a, c_a)
\]

\[

e_{\text{bob}} = \text{encrypt}(k_b, m_b | \text{seq}_b) \\
h_b = \text{MAC}(k_b, c_b)
\]

\[
\text{MAC}(k_b, c_b) = h_b \? \\
m_b | \text{seq}_b = \text{decrypt}(k_b, c_b)
\]

**property:** given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

\[
\text{MAC}(\text{key}, \text{message}) \rightarrow \text{token} \\
\text{MAC}(34\text{fbcbd1}, \text{“hello, world”}) = 0x59\text{cccc95723737f777e62bc756c8da5c}
\]

**property:** given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

**problem:** how do the parties know the keys?
policy: provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity**
(adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[
x \mod y \text{ is the remainder when } x \\
\text{is divided by } y
\]
e.g., \(10 \mod 8 = 2\); \(23 \mod 10 = 3\)
**policy**: provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model**: adversary can observe network data, tamper with packets, and insert its own packets

$x \mod y$ is the remainder when $x$ is divided by $y$
eq, 10 \mod 8 = 2; 23 \mod 10 = 3$

**known to everyone**: $p$ (prime), $g$
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[
x \mod y \text{ is the remainder when } x \text{ is divided by } y
\]
e.g., 10 mod 8 = 2; 23 mod 10 = 3

**known to everyone:** \( p \) (prime), \( g \)

\( g \) and \( p \) are related mathematically (\( g \) is a "primitive root" mod \( p \)). This relationship makes the next property possible.
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

x mod y is the remainder when x is divided by y
e.g., 10 mod 8 = 2; 23 mod 10 = 3

---

**known to everyone:** p (prime), g

g and p are related mathematically (g is a "primitive root" mod p). this relationship makes the next property possible.

---

**property:** given $g^r \mod p$, it is (virtually) impossible to determine r even if you know g and p
policy: provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**known to everyone:** $p$ (prime), $g$

g and $p$ are related mathematically ($g$ is a "primitive root" mod $p$). This relationship makes the next property possible.

---

**property:** given $g^r \mod p$, it is (virtually) impossible to determine $r$ *even if* you know $g$ and $p$

---

$x \mod y$ is the remainder when $x$ is divided by $y$
e.g., 10 mod 8 = 2; 23 mod 10 = 3

---

 alice

 bob
**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**known to everyone:** $p$ (prime), $g$

$g$ and $p$ are related mathematically ($g$ is a "primitive root" mod $p$). This relationship makes the next property possible.

**property:** given $g^r \mod p$, it is (virtually) impossible to determine $r$ even if you know $g$ and $p$

**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

$x \mod y$ is the remainder when $x$ is divided by $y$
e.g., $10 \mod 8 = 2$; $23 \mod 10 = 3$

**alice**
pick random $a$

**bob**
pick random $b$
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[
x \mod y \text{ is the remainder when } x \text{ is divided by } y
\]
e.g., \(10 \mod 8 = 2\); \(23 \mod 10 = 3\)

**known to everyone:** \(p\) (prime), \(g\)

g and \(p\) are related mathematically (\(g\) is a “primitive root” mod \(p\)). This relationship makes the next property possible.

**property:** given \(g^r \mod p\), it is (virtually) impossible to determine \(r\) even if you know \(g\) and \(p\)

**alice**
pick random \(a\)

\[
a g^a \mod p
\]

**bob**
pick random \(b\)

\[
b g^b \mod p
\]
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**known to everyone:** $p$ (prime), $g$

$g$ and $p$ are related mathematically ($g$ is a "primitive root" mod $p$). This relationship makes the next property possible.

---

**property:** given $g^r \mod p$, it is (virtually) impossible to determine $r$ even if you know $g$ and $p$

---

$x \mod y$ is the remainder when $x$ is divided by $y$.

- $10 \mod 8 = 2$
- $23 \mod 10 = 3$

---

**alice**

pick random $a$

$g^a \mod p$

calculate $(g^b \mod p)^a \mod p$

**bob**

pick random $b$

$g^b \mod p$

calculate $(g^a \mod p)^b \mod p$

---

Katrina LaCurts | lacurts@mit.edu | 6.1800 2024
**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**policy:** 
- **confidentiality** (adversary cannot learn message contents)
- **integrity** (adversary cannot tamper with packets and go undetected)

**known to everyone:** 
- $p$ (prime), $g$

$g$ and $p$ are related mathematically ($g$ is a "primitive root" mod $p$). This relationship makes the next property possible.

**property:** given $g^r \mod p$, it is (virtually) impossible to determine $r$ even if you know $g$ and $p$

---

**alice**
- pick random $a$
- $g^a \mod p$
- calculate $(g^b \mod p)^a \mod p$

**bob**
- pick random $b$
- $g^b \mod p$
- calculate $(g^a \mod p)^b \mod p$

**key =** $g^{ab} \mod p$
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

\[
x \mod y \text{ is the remainder when } x \text{ is divided by } y
\]

e.g., \(10 \mod 8 = 2; 23 \mod 10 = 3\)

**known to everyone:** \(p\) (prime), \(g\)

\(g\) and \(p\) are related mathematically (\(g\) is a "primitive root" mod \(p\)). This relationship makes the next property possible.

**property:** given \(g^r \mod p\), it is (virtually) impossible to determine \(r\) even if you know \(g\) and \(p\)
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**x mod y** is the remainder when **x** is divided by **y**

- e.g., 10 mod 8 = 2; 23 mod 10 = 3

---

**known to everyone:** **p** (prime), **g**

- **g** and **p** are related mathematically (**g** is a "primitive root" mod **p**). This relationship makes the next property possible.

---

**property:** given **g^r mod p**, it is (virtually) impossible to determine **r** even if you know **g** and **p**

---

Alice and Bob:

- **Alice**:
  - pick random **a**
  - calculate **(g^b mod p)^a mod p**
  - key = **g^ab mod p**

- **Bob**:
  - pick random **b**
  - calculate **(g^a mod p)^b mod p**

An observer on the network knows **p**, **g**, **g^a mod p**, and **g^b mod p**, but cannot use that information to learn **a** or **b**

and thus cannot calculate the key.
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**x mod y** is the remainder when x is divided by y

- e.g., 10 mod 8 = 2; 23 mod 10 = 3

---

**known to everyone:** p (prime), g

- g and p are related mathematically (g is a "primitive root" mod p). this relationship makes the next property possible.

---

**property:** given \( g^r \mod p \), it is (virtually) impossible to determine r even if you know g and p

---

**alice**

- pick random a

---

**bob**

- pick random b
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**x mod y** is the remainder when *x* is divided by *y*

E.g., 10 mod 8 = 2; 23 mod 10 = 3

---

**known to everyone:** *p* (prime), *g*

*g* and *p* are related mathematically (*g* is a "primitive root" mod *p*). This relationship makes the next property possible.

---

**property:** given *g^r mod p*, it is (virtually) impossible to determine *r* even if you know *g* and *p*

---

**alice**

pick random *a*

**eve**

pick random *e*

**bob**

pick random *b*
policy: provide confidentiality (adversary cannot learn message contents) and integrity (adversary cannot tamper with packets and go undetected)

threat model: adversary can observe network data, tamper with packets, and insert its own packets

\[ x \mod y \] is the remainder when \( x \) is divided by \( y \)

- e.g., 10 mod 8 = 2; 23 mod 10 = 3

known to everyone: \( p \) (prime), \( g \)

- \( g \) and \( p \) are related mathematically (\( g \) is a “primitive root” mod \( p \)). This relationship makes the next property possible.

property: given \( g^r \mod p \), it is (virtually) impossible to determine \( r \) even if you know \( g \) and \( p \)
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**known to everyone:** $p$ (prime), $g$

$g$ and $p$ are related mathematically ($g$ is a "primitive root" mod $p$). This relationship makes the next property possible.

**property:** given $g^r \mod p$, it is (virtually) impossible to determine $r$ even if you know $g$ and $p$

---

**$x \mod y$** is the remainder when $x$ is divided by $y$

e.g., $10 \mod 8 = 2$; $23 \mod 10 = 3$

---

**alice**

pick random $a$

$g^a \mod p$

---

**eve**

pick random $e$

$g^e \mod p$

$g^e \mod p$

---

**bob**

pick random $b$

$g^b \mod p$

$g^b \mod p$
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**x mod y** is the remainder when `x` is divided by `y`
- e.g., `10 mod 8 = 2`; `23 mod 10 = 3`

**known to everyone:** `p` (prime), `g`
- `g` and `p` are related mathematically (g is a "primitive root" mod p). This relationship makes the next property possible.

**property:** given `g^r mod p`, it is (virtually) impossible to determine `r` even if you know `g` and `p`

---

<table>
<thead>
<tr>
<th>alice</th>
<th>eve</th>
<th>bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>pick random <code>a</code></td>
<td>pick random <code>e</code></td>
<td>pick random <code>b</code></td>
</tr>
<tr>
<td><code>g^a mod p</code></td>
<td><code>g^e mod p</code></td>
<td><code>g^b mod p</code></td>
</tr>
<tr>
<td><code>g^e mod p</code></td>
<td><code>g^e mod p</code></td>
<td><code>g^e mod p</code></td>
</tr>
</tbody>
</table>

- `k_1 = (g^e)^a mod p`
- `k_2 = (g^e)^b mod p`
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

x mod y is the remainder when x is divided by y
e.g., 10 mod 8 = 2; 23 mod 10 = 3

**known to everyone:** p (prime), g
g and p are related mathematically (g is a "primitive root" mod p). this relationship makes the next property possible.

**property:** given $g^r \mod p$, it is (virtually) impossible to determine r even if you know g and p

---

alice
- pick random a
- $g^a \mod p$
- $g^e \mod p$
- $k_1 = (g^e)^a \mod p$

eve
- pick random e
- $g^b \mod p$
- $g^e \mod p$

bob
- pick random b
- $k_2 = (g^e)^b \mod p$

eve can calculate $k_1$ and $k_2$
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**x mod y** is the remainder when x is divided by y

- e.g., 10 mod 8 = 2; 23 mod 10 = 3

---

**known to everyone:** p (prime), g

- g and p are related mathematically (g is a “primitive root” mod p). This relationship makes the next property possible.

---

**property:** given $g^r \text{ mod } p$, it is (virtually) impossible to determine r even if you know g and p

---

**alice**

- pick random a

**eve**

- pick random e

**bob**

- pick random b

\[
g^a \text{ mod } p \quad \rightarrow \quad g^e \text{ mod } p
\]

\[
g^e \text{ mod } p \quad \rightarrow \quad g^e \text{ mod } p
\]

\[
k_1 = (g^e)^a \text{ mod } p
\]

\[
k_2 = (g^e)^b \text{ mod } p
\]

Eve can calculate $k_1$ and $k_2$

- encrypt($k_1$, m)
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**known to everyone:** $p$ (prime), $g$

g and $p$ are related mathematically ($g$ is a "primitive root" mod $p$). This relationship makes the next property possible.

---

**property:** given $g^r \mod p$, it is (virtually) impossible to determine $r$ even if you know $g$ and $p$
**policy:** provide confidentiality (adversary cannot learn message contents) and integrity (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**known to everyone:** $p$ (prime), $g$

$g$ and $p$ are related mathematically ($g$ is a “primitive root” mod $p$). This relationship makes the next property possible.

**property:** given $g^r \mod p$, it is (virtually) impossible to determine $r$ even if you know $g$ and $p$

---

$x \mod y$ is the remainder when $x$ is divided by $y$
e.g., $10 \mod 8 = 2$; $23 \mod 10 = 3$

---

**alice**

pick random $a$

$g^a \mod p$

$g^e \mod p$

$k_1 = (g^e)^a \mod p$

**eve**

pick random $e$

$g^b \mod p$

$g^e \mod p$

**bob**

pick random $b$

$g^b \mod p$

$g^e \mod p$

$k_2 = (g^e)^b \mod p$

**eve can calculate**

$k_1$ and $k_2$

decrypt $m$

encrypt ($k_1$, $m$)
**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

\[ x \mod y \text{ is the remainder when } x \text{ is divided by } y \]
\[ \text{e.g., } 10 \mod 8 = 2; \ 23 \mod 10 = 3 \]

**known to everyone:** \( p \) (prime), \( g \)
\( g \) and \( p \) are related mathematically (\( g \) is a "primitive root" mod \( p \)). This relationship makes the next property possible.

**property:** given \( g^r \mod p \), it is (virtually) impossible to determine \( r \) even if you know \( g \) and \( p \)

**problem:** alice and bob don’t know they’re not communicating directly

\[
\begin{align*}
\text{alice} & \quad \text{pick random } a \\
& \quad \text{pick random } \ e \\
& \quad \text{pick random } \ b \\
\rightarrow & \quad g^a \mod p \\
\rightarrow & \quad g^e \mod p \\
\rightarrow & \quad g^e \mod p \\
& \quad \quad \quad \quad \quad \quad \quad \text{eve can calculate } k_1 \text{ and } k_2 \\
& \quad \quad \quad \quad \quad \quad \quad k_1 = (g^e)^a \mod p \\
& \quad \quad \quad \quad \quad \quad \quad k_2 = (g^e)^b \mod p \\
\rightarrow & \quad \text{encrypt}(k_1, m) \\
\rightarrow & \quad \text{decrypt } m \\
& \quad \quad \quad \quad \quad \quad \quad \text{encrypt}(k_2, m) \\
\end{align*}
\]
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

\{public_key, secret_key\}
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

Users generate key pairs; the two keys in the pair are related mathematically

\{public_key, secret_key\}

\begin{align*}
\text{sign}(\text{secret_key, message}) & \rightarrow \text{sig} \\
\text{verify}(\text{public_key, message, sig}) & \rightarrow \text{yes/no}
\end{align*}
Threat model: adversary can observe network data, tamper with packets, and insert its own packets.

Cryptographic signatures allow users to verify identities using public-key cryptography.

Users generate key pairs; the two keys in the pair are related mathematically:

\{public_key, secret_key\}

\[\text{sign}(\text{secret_key}, \text{message}) \rightarrow \text{sig}\]

\[\text{verify}(\text{public_key}, \text{message}, \text{sig}) \rightarrow \text{yes/no}\]

Property: it is (virtually) impossible to compute \text{sig} without \text{secret_key}.

Policy: provide confidentiality (adversary cannot learn message contents) and integrity (adversary cannot tamper with packets and go undetected).
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

\[ \{\text{public_key}, \text{secret_key}\} \]

\[ \text{sign} (\text{secret_key}, \text{message}) \rightarrow \text{sig} \]

\[ \text{verify} (\text{public_key}, \text{message}, \text{sig}) \rightarrow \text{yes}/\text{no} \]

**property:** it is (virtually) impossible to compute \( \text{sig} \) without \( \text{secret_key} \)
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

Users generate **key pairs**; the two keys in the pair are related mathematically

\{public_key, secret_key\}

**sign**\( (secret_key, message) \rightarrow sig \)

**verify**\( (public_key, message, sig) \rightarrow yes/no \)

**property:** it is (virtually) impossible to compute \( sig \) without \( secret_key \)
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**cryptographic signatures** allow users to verify identities using public-key cryptography

Users generate **key pairs**; the two keys in the pair are related mathematically:

\[
\{ \text{public	extunderscore key}, \text{secret	extunderscore key} \}
\]

**sign** (secret_key, message) → sig

**verify** (public_key, message, sig) → yes/no

**property:** it is (virtually) impossible to compute sig without secret_key

---

**alice**

\[ m = \text{original message} \]

\[ c = \text{encrypt}(k_a, m \mid \text{seq}_a) \]

\[ h = \text{MAC}(k_a, c) \]

\[ \text{sig} = \text{sign}(\text{secret\textunderscore key}_a, m \mid \text{seq}_a) \]

**bob**

\[ c \mid h \mid \text{sig} \]
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

\{public_key, secret_key\}

\[\text{sign}(\text{secret_key}, \text{message}) \rightarrow \text{sig} \]

\[\text{verify}(\text{public_key}, \text{message}, \text{sig}) \rightarrow \text{yes/no}\]

**property:** it is (virtually) impossible to compute \(\text{sig}\) without \(\text{secret_key}\)
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

Users generate **key pairs**; the two keys in the pair are related mathematically

\{public_key, secret_key\}

\[
\text{sign}(\text{secret_key}, \text{message}) \rightarrow \text{sig} \\
\text{verify}(\text{public_key}, \text{message}, \text{sig}) \rightarrow \text{yes/no}
\]

**property:** it is (virtually) impossible to compute \(\text{sig}\) without \(\text{secret_key}\)

This is a **rough outline** of how to think about public signatures in the context of this lecture. In reality, things work a bit differently; you'll see an example in a few minutes.
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

<table>
<thead>
<tr>
<th>cryptographic signatures</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>allow users to verify identities using public-key cryptography</td>
<td></td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>users generate <strong>key pairs</strong>; the two keys in the pair are related mathematically</td>
</tr>
<tr>
<td>{public_key, secret_key}</td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sign(secret_key, message) → sig</td>
</tr>
<tr>
<td>verify(public_key, message, sig) → yes/no</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>alice</th>
<th>bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>m = original message</td>
<td></td>
</tr>
<tr>
<td>c = encrypt(k_a, m</td>
<td>seq_a)</td>
</tr>
<tr>
<td>h = MAC(k_a, c)</td>
<td></td>
</tr>
<tr>
<td>sig = sign(secret_key_a, m</td>
<td>seq_a)</td>
</tr>
</tbody>
</table>

```
c | h | sig
```

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC(k_a, c) == h ?</td>
</tr>
<tr>
<td>m</td>
</tr>
<tr>
<td>verify(m</td>
</tr>
</tbody>
</table>

**property:** it is (virtually) impossible to compute sig without secret\_key

`this is a rough outline of how to think about public signatures in the context of this lecture. in reality, things work a bit differently; you’ll see an example in a few minutes`

**how do we distribute public keys?**
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

<table>
<thead>
<tr>
<th>cryptographic signatures</th>
<th>allow users to verify identities using public-key cryptography</th>
</tr>
</thead>
<tbody>
<tr>
<td>users generate <strong>key pairs</strong>; the two keys in the pair are related mathematically</td>
<td>{public_key, secret_key}</td>
</tr>
<tr>
<td>sign(secret_key, message) → sig</td>
<td>verify(public_key, message, sig) → yes/no</td>
</tr>
<tr>
<td>property: it is (virtually) impossible to compute sig without secret_key</td>
<td></td>
</tr>
</tbody>
</table>

**how do we distribute public keys?**
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

\{public_key, secret_key\}

**sign**\( (\text{secret_key}, \text{message}) \rightarrow \text{sig} \)

**verify**\( (\text{public_key}, \text{message}, \text{sig}) \rightarrow \text{yes/no} \)

**property:** it is (virtually) impossible to compute \( \text{sig} \) without \text{secret_key} \n
\( x_{pk} = x \)'s public key

\( x_{sk} = x \)'s secret key (known only to \( x \))

**how do we distribute public keys?**
**Policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**Threat Model:** adversary can observe network data, tamper with packets, and insert its own packets

**Cryptographic Signatures** allow users to verify identities using public-key cryptography

Users generate **key pairs**; the two keys in the pair are related mathematically

\{public_key, secret_key\}

**Sign**\( (\text{secret_key}, \text{message}) \rightarrow \text{sig} \)

**Verify**\( (\text{public_key}, \text{message}, \text{sig}) \rightarrow \text{yes/no} \)

**Property:** it is (virtually) impossible to compute \( \text{sig} \) without \( \text{secret_key} \)

**How do we distribute public keys?**

\( x_{pk} = x\text{'s public key} \)

\( x_{sk} = x\text{'s secret key (known only to } x \text{)} \)
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity**
(adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**cryptographic signatures** allow users to verify identities using public-key cryptography

Users generate **key pairs;** the two keys in the pair are related mathematically

\{public\_key, secret\_key\}

**sign**(secret\_key, message) → sig

**verify**(public\_key, message, sig) → yes/no

**property:** it is (virtually) impossible to compute sig without secret\_key

\[xpk = x's \text{ public key}\]

\[xsk = x's \text{ secret key (known only to } x)\]

---

how do we distribute public keys?
policy: provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

cryptographic signatures allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

\{public\_key, secret\_key\}

sign(secret\_key, message) → sig
verify(public\_key, message, sig) → yes/no

**property:** it is (virtually) impossible to compute sig without secret\_key

---

alice: alice\_pk
bob: bob\_pk
...
server: server\_pk

alice and bob could ask the server for any public keys they need, but that doesn’t scale, and we also have to figure out how to distribute the server’s public key

\[x_{pk} = x’s\ public\ key\]
\[x_{sk} = x’s\ secret\ key\ (known\ only\ to\ x)\]

---

**how do we distribute public keys?**
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

\{public_key, secret_key\}

**sign**(secret_key, message) → sig
**verify**(public_key, message, sig) → yes/no

**property:** it is (virtually) impossible to compute sig without secret_key
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

Users generate **key pairs**; the two keys in the pair are related mathematically

\{public_key, secret_key\}

**sign(secret_key, message) → sig**

**verify(public_key, message, sig) → yes/no**

**property:** it is (virtually) impossible to compute **sig** without **secret_key**
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

\[
\{\text{public\_key}, \text{secret\_key}\}
\]

\[
\text{sign} (\text{secret\_key}, \text{message}) \rightarrow \text{sig}
\]

\[
\text{verify} (\text{public\_key}, \text{message}, \text{sig}) \rightarrow \text{yes/no}
\]

**property:** it is (virtually) impossible to compute \(\text{sig}\) without \(\text{secret\_key}\)

server pre-computes **signed** messages that map names to their public keys

\[
\text{sign} (\text{server}_s, \text{“alice: alice}_{pk}\text{”}) \rightarrow \text{sig}
\]
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

- **cryptographic signatures** allow users to verify identities using public-key cryptography

  - Users generate **key pairs**; the two keys in the pair are related mathematically
    - \{public_key, secret_key\}

  - **sign\((\text{secret_key}, \text{message})\) \rightarrow \text{sig}**

  - **verify\((\text{public_key}, \text{message}, \text{sig})\) \rightarrow \text{yes/no}**

- **property:** it is (virtually) impossible to compute \text{sig} without \text{secret_key}

- Server pre-computes **signed** messages that map names to their public keys

  - **\text{sign}(server_{sk}, \text{“alice: alice}_{pk}\text{”) \rightarrow sig}**

  - **alice, alice}_{pk}, \text{sig}**
policy: provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

\{public-key, secret-key\}

**sign**\((\text{secret-key}, \text{message}) \rightarrow \text{sig}\)

**verify**\((\text{public-key}, \text{message}, \text{sig}) \rightarrow \text{yes/no}\)

**property:** it is (virtually) impossible to compute \text{sig} without \text{secret-key}

\begin{align*}
\text{alice} & : \text{alice}_{sk} \\
\text{bob} & : \text{bob}_{sk} \\
\text{server} & : \text{server}_{sk} \\
\text{server}_{pk} & \text{pre-computes signed messages that map names to their public keys} \\
\text{sign(} & \text{server}_{sk}, \text{“alice: alice}_{pk}”) \rightarrow \text{sig} \\
& \text{alice, alice}_{pk}, \text{sig} \\
& \text{anyone can verify that the authority signed this message given server}_{pk}, \text{but the server itself doesn’t have to distribute the signed messages}
\end{align*}
**Policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**Threat Model:** adversary can observe network data, tamper with packets, and insert its own packets

**Cryptographic Signatures:** allow users to verify identities using public-key cryptography

Users generate **key pairs**; the two keys in the pair are related mathematically:

\[
\{ \text{public_key}, \text{secret_key} \}\]

\[
\text{sign(}\text{secret_key}, \text{message}) \rightarrow \text{sig}
\]

\[
\text{verify(}\text{public_key}, \text{message}, \text{sig}) \rightarrow \text{yes/no}
\]

**Property:** it is (virtually) impossible to compute \text{sig} without \text{secret_key}

**Certificate Authority**

Server pre-computes **signed** messages that map names to their public keys

\[
\text{sign(}\text{server}_{sk}, \text{“alice: alice}_{pk}\text{”}) \rightarrow \text{sig}
\]

\[
\text{alice, alice}_{pk}, \text{sig}
\]

Anyone can verify that the authority signed this message given \text{server}_{pk}, but the server itself doesn’t have to distribute the signed messages.
policy: provide confidentiality (adversary cannot learn message contents) and integrity (adversary cannot tamper with packets and go undetected)

threat model: adversary can observe network data, tamper with packets, and insert its own packets

cryptographic signatures allow users to verify identities using public-key cryptography

users generate key pairs; the two keys in the pair are related mathematically

\{public_key, secret_key\}

\[
\text{sign}(\text{secret_key}, \text{message}) \rightarrow \text{sig}\\
\text{verify}(\text{public_key}, \text{message}, \text{sig}) \rightarrow \text{yes/no}
\]

property: it is (virtually) impossible to compute \text{sig} without \text{secret_key}
**Policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity**
(adversary cannot tamper with packets and go undetected)

**Threat Model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**Cryptographic Signatures** allow users to verify identities using public-key cryptography

Users generate **key pairs**; the two keys in the pair are related mathematically

\[
\{\text{public}_\text{key}, \text{secret}_\text{key}\}
\]

**Sign** (secret_key, message) → sig
**Verify** (public_key, message, sig) → yes/no

**Property:** it is (virtually) impossible to compute sig without secret_key
policy: provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

\{**public_key**, **secret_key**\}

\[\text{sign}(**secret_key**, \text{message}) \rightarrow \text{sig}\]
\[\text{verify}(**public_key**, \text{message}, \text{sig}) \rightarrow \text{yes/no}\]

**property:** it is (virtually) impossible to compute **sig** without **secret_key**
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

\{**public_key**, **secret_key**\}

**sign**(secret_key, message) → sig
**verify**(public_key, message, sig) → yes/no

**property:** it is (virtually) impossible to compute sig without secret_key

---

**TLS handshake**

---

client

| ClientHello {version, seq, session_id, cipher suites, compression func} |
| ServerHello {version, seq, session_id, cipher suite, compression func} |
| {server certificate, CA certificates} |
| ServerHelloDone |

server

client verifies authenticity of server
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

\[ \{\text{public_key}, \text{secret_key}\} \]

**sign**\((\text{secret_key}, \text{message}) \to \text{sig}\)**

**verify**\((\text{public_key}, \text{message}, \text{sig}) \to \text{yes/no}\)

**property:** it is (virtually) impossible to compute **sig** without **secret_key**

---

**TLS handshake**

client

- **ClientHello** \(\{\text{version, seq, session_id, cipher suites, compression func}\}\)

- **ServerHello** \(\{\text{version, seq, session_id, cipher suite, compression func}\}\)

  - \(\text{server certificate, CA certificates}\)

- **ServerHelloDone**

server

client verifies authenticity of server

- **ClientKeyExchange** \(\{\text{encrypt(server_pub_key, pre_master_secret)}\}\)
**Threat Model:** 
An adversary can observe network data, tamper with packets, and insert its own packets.

**Policy:** 
Provide confidentiality (adversary cannot learn message contents) and integrity (adversary cannot tamper with packets and go undetected).

**Cryptographic Signatures:**
Allow users to verify identities using public-key cryptography.

Users generate **key pairs**: the two keys in the pair are related mathematically:

\{public_key, secret_key\}

**Sign:**
\text{sign}(secret_key, message) \rightarrow \text{sig}

**Verify:**
\text{verify}(public_key, message, sig) \rightarrow yes/no

**Property:**
It is (virtually) impossible to compute sig without secret_key.

---

**TLS Handshake**

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClientHello</strong> {version, seq, session_id, cipher suites, compression func}</td>
<td>{server certificate, CA certificates}</td>
</tr>
<tr>
<td><strong>ServerHello</strong> {version, seq, session_id, cipher suite, compression func}</td>
<td>{server certificate, CA certificates}</td>
</tr>
<tr>
<td><strong>ServerHelloDone</strong></td>
<td></td>
</tr>
</tbody>
</table>

Client verifies authenticity of server:

**ClientKeyExchange** \{\text{encrypt}(server_pub_key, \text{pre_master_secret})\}

**Compute:**

\[
\text{master_secret} = \text{PRF} (\text{pre_master_secret}, \text{“master secret”}, \text{seq} | \text{seq})
\]

\[
\text{key_block} = \text{PRF} (\text{master_secret}, \text{“key expansion”}, \text{seq} | \text{seq})
\]

= \{client_MAC_key, server_MAC_key, client_encrypt_key, server_encrypt_key, ...\}
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

\[
\{\text{public}\_\text{key}, \text{secret}\_\text{key}\}\]

**sign**(secret_key, message) → sig

**verify**(public_key, message, sig) → yes/no

**property:** it is (virtually) impossible to compute sig without secret_key

---

**TLS handshake**

**client**

ClientHello \{version, seq, session_id, cipher suites, compression func\}

ServerHello \{version, seq, session_id, cipher suite, compression func\}

\{server certificate, CA certificates\}

ServerHelloDone

client verifies authenticity of server

ClientKeyExchange \{encrypt(server_pub_key, pre_master_secret)\}

compute

\[
\text{master}\_\text{secret} = \text{PRF}(\text{pre_master}\_\text{secret}, \text{“master secret”}, \text{seq} \mid \text{seq})
\]

key_block = PRF(master_secret, “key expansion”, seq | seq) = \{client_MAC_key, server_MAC_key, client_encrypt_key, server_encrypt_key, ...\}

Finished \{sign(client_MAC_key, encrypt(client_encrypt_key, MAC(master_secret, previous_messages)))\}

**server**
**policy:** provide **confidentiality** (adversary cannot learn message contents) and **integrity** (adversary cannot tamper with packets and go undetected)

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets

---

**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

{public_key, secret_key}

**sign**(secret_key, message) → sig

**verify**(public_key, message, sig) → yes/no

**property:** it is (virtually) impossible to compute sig without secret_key

---

**client**

ClientHello {version, seq, session_id, cipher suites, compression func}

ServerHello {version, seq, session_id, cipher suite, compression func}

{server certificate, CA certificates}

ServerHelloDone

ClientKeyExchange {encrypt(server_pub_key, pre_master_secret)}

compute

master_secret = PRF(pre_master_secret, “master secret”, seq | seq)
key_block = PRF(master_secret, “key expansion”, seq | seq)
= {client_MAC_key,
server_MAC_key,
client_encrypt_key,
server_encrypt_key,
...}

Finished {sign(client_MAC_key, encrypt(client_encrypt_key,
MAC(master_secret, previous_messages)))}

Finished {sign(server_MAC_key, encrypt(server_encrypt_key,
MAC(master_secret, previous_messages)))}

**server**
ECDSA signatures rely on a pseudo-random number, typically notated as K, that’s used to derive two additional numbers, R and S. To verify a signature as valid, a party must check the equation involving R and S, the signer’s public key, and a cryptographic hash of the message. When both sides of the equation are equal, the signature is valid.

In a writeup published Wednesday, security firm Sophos further explained the process:

```
S1. Select a cryptographically sound random integer K between 1 and N-1 inclusive.
S2. Compute R from K using Elliptic Curve multiplication.
S3. In the unlikely event that R is zero, go back to step 1 and start over.
S4. Compute S from K, R, the hash to be signed, and the private key.
S5. In the unlikely event that S is zero, go back to step 1 and start over.
```
Madden wrote:

“Guess which check Java forgot?

That's right. Java’s implementation of ECDSA signature verification didn’t check if R or S were zero, so you could produce a signature value in which they are both 0 (appropriately encoded) and Java would accept it as a valid signature for any message and for any public key. The digital equivalent of a blank ID card."
**encryption** provides confidentiality

Here, we are using symmetric-key encryption: the same key is used to encrypt and decrypt

\[
\text{encrypt(key, message)} \rightarrow \text{ciphertext}
\]

\[
\text{decrypt(key, ciphertext)} \rightarrow \text{message}
\]

\[
\text{encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c}
\]

\[
\text{decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = “hello, world”}
\]

**property:** given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key

**MACs** provides integrity

\[
\text{MAC(key, message)} \rightarrow \text{token}
\]

\[
\text{MAC(34fbcbd1, “hello, world”) = 0x59cccc95723737f777e62bc756c8da5c}
\]

**property:** given the message, it is (virtually) impossible to obtain the token without knowing the key

It is also impossible to go in the reverse direction: given token, you can’t get message even with the key

in the next lecture, we are going to use a different style of encryption — public-key encryption — to provide confidentiality in a different system

**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

\[
\{\text{public_key}, \text{secret_key}\}
\]

\[
\text{sign(secret_key, message)} \rightarrow \text{sig}
\]

\[
\text{verify(public_key, message, sig)} \rightarrow \text{yes/no}
\]

**property:** it is (virtually) impossible to compute sig without secret_key
**secure channels** protect us from adversaries that can observe and tamper with packets in the network.

Encrypting with **symmetric keys** provides confidentiality, and using **MACs** provides integrity. **Diffie-Hellman key exchange** lets us exchange the symmetric key securely.

to verify identities, we use **public-key cryptography** and cryptographic **signatures**. We often distribute public keys via **certificate authorities**, though this method is not perfect.