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
this week, we’re going to turn to adversaries that are observing data on the network

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

```plaintext
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 4000 ........EH.S..@.
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  \..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

```
14:10:28.658392 331061605us tsft -98dB noise antenna 1 5785 MHz 11a ht/20 [bit 20] +QoS IP 18.4.86.46.80 > 18.21.134.133.59071: Flags [], seq 9009:10457, ack 1, win 238, options [nop,nop,TS val 1469784939 ecr 1030694527], length 1448: HTTP
0x0040:  0d0a 0a09 0909 3c6f 7074 696f 6e20 7661  ......<option.va
0x0050:  6c75 653d 2234 3439 223e 266e 6273703b  lue="449">nbsp;
0x0060:  2026 6e62 7370 3b54 6f77 6e20 53717561  .nbsp;Town.Squa
0x0070:  7265 3c2f 6f7074696f6e3e0a090909093c6f  re</option>....
0x0080:  7074 696f6e 7661 6c75 653d 2234 3430 223e  tion.value="440">D&amp;D.My
0x0090:  3e26 6e6273703b 2026 6e6273703b50 6c61  >&nbsp;nbsp;Mons
0x00a0:  7965 6f6e 204d 6173746572277320477569  geon.Master's.Gu
0x00b0:  653c 2f6f7074696f6e3e0a090909093c6f  ide</option>....
```
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

principals
(identifies client on server)

request

server

0x0000: aaaa 0300 0000 0800 4500 009b 2acb 0000 ........E...*
0x0010: ff11 d8b2 1215 c4c3 e000 00fb 14e9 14e9 .............
0x0020: 0087 a623 0000 0000 002 0000 0000 0001 .#...........
0x0030: 184d 6174 74e2 8099 732 4d61 6342 6f6f .MacBook.
0x0040: 6b20 4169 7220 283 290f 5f63 6f6d 7061 .k.Air.(3).companion.
0x0050: 6e69 6f6e 2d6c 696e 6b04 5f74 6370 056c .nion-link.
0x0060: 6f6c 616c 0000 1000 0116 5468 6f6d 6173 .local......
0x0070: e280 9973 204d 6163 2050 726f 0f5f 636f 7320 .s.Mac.P.
0x0080: 6d61 6342 6f6f 6b20 4169 7220 283 29c0 1e00 00fb 14e9 14e9 .s.Book.Air.(3).Z.
0x0090: 726f 6861 6e e280 99 73 204d 6163 2050 726f 0f5f 636f 7061 .Pro.(2)....
0x00a0: 6e69 6f6e 2d6c 696e 6b04 5f74 6370 056c .nion-link.
0x00b0: 6f6c 616c 0000 1000 0116 5468 6f6d 6173 .local......
0x00c0: e280 9973 204d 6163 2050 726f 0f5f 636f 7320 .s.Mac.P.
0x00d0: 6d61 6342 6f6f 6b20 4169 7220 283 29c0 1e00 00fb 14e9 14e9 .s.Book.Air.(3).Z.
0x00e0: 726f 6861 6e e280 99 73 204d 6163 2050 726f 0f5f 636f 7061 .Pro.(2)....
0x00f0: 6e69 6f6e 2d6c 696e 6b04 5f74 6370 056c .nion-link.
0x0100: 6f6c 616c 0000 1000 0116 5468 6f6d 6173 .local......
0x0110: e280 9973 204d 6163 2050 726f 0f5f 636f 7320 .s.Mac.P.
0x0120: 6d61 6342 6f6f 6b20 4169 7220 283 29c0 1e00 00fb 14e9 14e9 .s.Book.Air.(3).Z.
0x0130: 726f 6861 6e e280 99 73 204d 6163 2050 726f 0f5f 636f 7061 .Pro.(2)....
0x0140: 6e69 6f6e 2d6c 696e 6b04 5f74 6370 056c .nion-link.
0x0150: 6f6c 616c 0000 1000 0116 5468 6f6d 6173 .local......
0x0160: e280 9973 204d 6163 2050 726f 0f5f 636f 7320 .s.Mac.P.
0x0170: 6d61 6342 6f6f 6b20 4169 7220 283 29c0 1e00 00fb 14e9 14e9 .s.Book.Air.(3).Z.
0x0180: 726f 6861 6e e280 99 73 204d 6163 2050 726f 0f5f 636f 7061 .Pro.(2)....
0x0190: 6e69 6f6e 2d6c 696e 6b04 5f74 6370 056c .nion-link.
0x01a0: 6f6c 616c 0000 1000 0116 5468 6f6d 6173 .local......
0x01b0: e280 9973 204d 6163 2050 726f 0f5f 636f 7320 .s.Mac.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

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

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

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**

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**

\[
\begin{align*}
\text{alice} & \quad \text{bob} \\
\text{c} = \text{encrypt}(k, m) & \quad \text{in practice, we’d use one key to encrypt and a different one to MAC} \\
h = \text{MAC}(k, c) & \\
\text{MAC}(k, c) \equiv h ? \\
m = \text{decrypt}(k, c)
\end{align*}
\]
**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

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

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

**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**

\[
\text{problem: replay attacks} \\
\text{adversary could intercept a message, re-send it at a later time}
\]
**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

\[
\text{MAC(key, message)} \rightarrow \text{token} \\
\text{MAC}(34fbcbd1, \text{"hello, world"}) = 0x59cccc95723737f777f62bc756c8da5c
\]

**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 the token, you can't get the message even with the key

**problem:** reflection attacks
adversary 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

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

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

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

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

**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

**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

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

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

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

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

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

\[
\text{known to everyone:}
\]

- **p** (prime), **g**

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

\[
\text{alice}
\]

- pick random **a**

- calculate \((g^a \mod p)^a \mod p\)

\[
\text{bob}
\]

- pick random **b**

- calculate \((g^b \mod p)^b \mod p\)

\[
\text{key} = g^{ab} \mod p
\]

\[
(g^x \mod p)^y \mod p = (g^y \mod p)^x \mod p = g^{xy} \mod p
\]

**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

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

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

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

\[
\begin{align*}
\text{alice} &\quad \text{eve} &\quad \text{bob} \\
pick\text{ random } a &\quad pick\text{ random } e &\quad \text{pick random } b \\
\quad g^a \mod p &\quad g^b \mod p &\quad \text{compute } k_1, k_2 \\
\quad g^e \mod p &\quad g^e \mod p &
\end{align*}
\]

\[
k_1 = (g^e)^a \mod p \\
k_2 = (g^e)^b \mod p
\]

\[
\text{eve can calculate } k_1 \text{ and } k_2
\]

\[
\begin{align*}
\text{encrypt}(k_1, m) &\rightarrow \\
\text{decrypt } m &\rightarrow \\
\text{encrypt}(k_2, m) &\rightarrow
\end{align*}
\]

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

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

**Verify** with **public_key**:

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

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

---

**alice**

\[
m = \text{original message} \\
c = \text{encrypt}(k_a, m \mid \text{seq}_a) \\
h = \text{MAC}(k_a, c) \\
\text{sig} = \text{sign}(\text{secret_key}_a, m \mid \text{seq}_a)
\]

**bob**

\[
\text{c} \mid \text{h} \mid \text{sig} \\
\text{MAC}(k_a, c) = \text{h}? \\
\text{m} \mid \text{seq}_a = \text{decrypt}(k_a, c) \\
\text{verify}(m \mid \text{seq}_a, \text{public_key}_a, \text{sig}) = \text{yes}?
\]

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

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}

alice: alice\_pk  

bob: bob\_pk  

server\_pk  

**how do we distribute public keys?**

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)
**Threat Model:** An 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: \(\text{sign(}\text{secret_key, message}\text{)} \rightarrow \text{sig}\)
- Verify: \(\text{verify(}\text{public_key, message, sig}\text{)} \rightarrow \text{yes/no}\)

**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).

**Certificate Authority:**

- **server** pre-computes signed messages that map names to their public keys: \(\{\text{alice, alice}_pk\}_{\text{server}_sk}\)
- 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.

- This server is a **certificate authority**, and that signed message is a **certificate**.
**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)

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

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

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

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

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

**Property:** it is (virtually) impossible to compute **sig** without **secret\_key**
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."
**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 distributed public keys via **certificate authorities**, though this method is not perfect.