In the news

Even so, the attack is just the latest episode in which hackers have gone after critical systems such as water plants, oil refineries, chemical plants or the electric grid — including a notorious incident in which Russia shut off part of Ukraine's power supply. It's also part of a growing plague involving ransomware, in which hackers demanding payments have crippled targets such as hospitals, police stations or municipal governments.

This could be the most serious successful attack the U.S. has faced yet.

Sen. Ed Markey (D-Ma.) said the federal government has long failed to devote the needed attention to pipeline security, and he pointed to a U.S. Government Accountability Office report that showed the TSA had only six full-time staff on pipeline security as recently as 2019.

"While we need more information about the circumstances that allowed the Colonial Pipeline cyberattack, we cannot ignore the longstanding inadequacies that allowed for, and enabled, cyber intrusions into our critical infrastructure," he said in a statement.
Biden Plans an Order to Strengthen Cyberdefenses. Will It Be Enough?

A hacking of a major pipeline, the latest evidence of the nation’s vulnerabilities to cyberattacks, prompted questions about whether the administration should go further.

And efforts to regulate minimum cybersecurity standards for companies that oversee critical systems have repeatedly failed, most notably in 2012, when lobbyists killed such an effort in Congress, arguing that the standards would be too expensive and too onerous for businesses.

“The ghost of 2012 hangs over this,” Mr. Lewis said. “But we’ve been recommending these same measures since there were two people on the internet.”

It would create a series of digital safety standards for federal agencies and contractors that develop software for the federal government, such as multifactor authentication, a version of what happens when consumers get a second code from a bank or credit-card company to allow them to log in. It would require federal agencies to take a “zero trust” approach to software vendors, granting them access to federal systems only when necessary, and require contractors to certify that they comply with steps to ensure that the software they deliver has not been infected with malware or does not contain exploitable vulnerabilities. And it would require that vulnerabilities in software be reported to the U.S. government.

Violators would risk having their products banned from sale to the federal government, which would, in essence, kill their viability in the commercial market.

The measures are intended to address the fact that the software company SolarWinds made for such an easy target for Russia’s premier intelligence agency, which used its software update to burrow into nine federal agencies as well as technology firms and even some utility companies. (Despite SolarWinds’ incredible access to federal networks, an intern had set the firm’s password to its software update mechanism to “SolarWinds123.”)
Lecture #22: 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

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.lt..Y_.V.....
0x0020: 61f8 18b2 4877 7f7b 8018 011d 835f 0000  a...Hw........`
0x0030: 0101 080a 8541 b29f 05aa ea3e 1503 0300  01...a3e 1503 0300
0x0040: 01040: 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

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. wednesday, we’ll talk about how you can protect meta-information (e.g., packet headers) from an adversary

principal
(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 0002 0000 0000 0001 ..........
0x0030: 184d 6174 74e2 8099 7320 4d61 6342 6f6f .MacBook.Air
0x0040: 6b20 4169 7220 2832 290f 5f63 6f6d 7061 .k.Air.(3)_.companion
0x0050: 6e69 6f6e 2d6c 696e 6b04 5f74 6370 056c companion._tcp.l
0x0060: e280 9973 204d 6163 2050 726f 2028 3229 .s.Mac.P
0x0070: 0f5f 636f 6d70 616e 696f 6e2d 6c69 .ro_companion-link
0x0080: 6e6b 045f 7463 7005 6c6f 6361 6c00 0010 .nk._tcp.local...
0x0090: 726f 2028 3329c0 1e00 1000 010d 4d61 7961 .Pro.(3)...
0x00a0: 8099 7320 4d61 6342 6f6f 6b20 4169 7220 .s.MacBook.Air
0x00b0: 2833 29c0 1e00 1000 010d 4d61 7961 e280 . padding
0x00c0: 9973 2069 5061 64c0 1e00 1000 010d 4d61 .s.iPad
0x00d0: 7961 e280 9973 2069 5061 64c0 1e00 1000 .iPad
0x00e0: 2833 29c0 1e00 1000 010d 4d61 7961 e280 . padding
0x00f0: 8400 1194 0012 0004 000e 0081 a641 ........Ag/h.Ag
**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
```

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

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

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

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

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

**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**

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

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

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

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

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

**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

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

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

\[
\begin{align*}
c | h
\end{align*}
\]

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

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

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

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

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

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

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

**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

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

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

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

**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**

\[
\text{it is also impossible to go in the reverse direction: given token, you can’t get message even with the key}
\]

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

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

**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

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

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

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

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

**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**

**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

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

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

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

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

**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)
Encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c

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

**Encrypted Text:**

**Property:** Given the ciphertext, it is (virtually) impossible to determine the message even if you know the key.

**MAC:**

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).
**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:** \( \text{sign}(\text{secret\_key}, \text{message}) \rightarrow \text{sig} \)

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

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

```
\begin{align*}
\text{alice} & \quad \text{bob} \\
\text{m} & = \text{original message} \\
\text{c} & = \text{encrypt}(k_a, m | seq_a) \\
\text{h} & = \text{MAC}(k_a, c) \\
\text{sig} & = \text{sign}(\text{secret\_key}_a, m | seq_a) \\
\hline \\
\text{c} & \quad \text{h} & \quad \text{sig} \\
\text{MAC}(k_a, c) & \Rightarrow \text{h} \quad ? \\
\text{m} | \text{seq}_a & = \text{decrypt}(k_a, c) \\
\text{verify}(m | seq_a, \text{public\_key}_a, \text{sig}) & \Rightarrow \text{yes} ?
\end{align*}
```

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

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

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

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

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

---

**alice**

\begin{align*}
\text{alice} & \quad \text{alice\_sk} \\
& \quad \text{bob: bob\_pk} \\
& \quad \ldots
\end{align*}

**bob**

\begin{align*}
& \quad \text{bob\_sk} \\
& \quad \text{server\_pk} \\
& \quad \text{server\_sk}
\end{align*}

**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

\[
x\_pk = x\’s \text{ public key} \\
x\_sk = x\’s \text{ secret key (known only to x)}
\]

**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

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

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

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

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

\[\text{server pre-computes signed messages that map names to their public keys}\]

\[\{\text{alice, alice_pk}\}_{\text{server_sk}}\]

\[\text{anyone can verify that the authority signed this message given server_pk, but the server itself doesn’t have to distribute the signed messages}\]

\[\text{this server is a certificate authority, and that signed message is a certificate}\]

Katrina LaCurts | lacurts@mit.edu | 6.033 2021
**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

client verifies authenticity of server

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**


Katrina LaCurts | lacurts@mit.edu | 6.033 2021
**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.