Lecture 12:
Encryption in Practice

6.5060 - Fall 2021
MIT
C-G, Devadas, Kalai,
Zeldovich

Plan

* File encryption
* Encrypted streams: TLS
* Encrypted messaging

Theme: Gap between properties that apps want & properties that standard schemes provide.

Logistics

- Lab 3 due 10/28
- Midterm 11/2

Recap: Encryption

* Weak (CPA-secure) enc, fixed-len msgs, shared key

Counter mode

* " " var-len msgs "

Enc-then-MAC

* Strong (CCA-secure) enc, " "

DH Key Exchange

* " " " " without a shared key

Today: Applications

Next time: Privacy/Crypto problems that encryption doesn't solve.

Surprise!?

→ With CRHFs, MACs, Signatures
AE, DH, PKE

you have the tools to understand essentially every widely used cryptographic protocol.*
(exception, RSA, lattices, blockchain, ...)

→ There really are not that many primitives in use in our systems.

BUT: As you'll see, the design/specs are still very complicated.

Why?

↳ Extra security & functionality properties

↳ Less often, but sometimes: Sloppy design

↳ Also, you'll see rules violated → often attacks

File Encryption

↳ Essentially what we've already seen.

↳ Bottom line: Use authenticated encryption AES-GCM

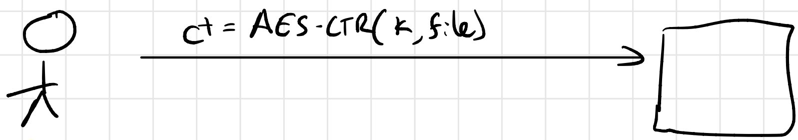
Example: **WhatsApp Encrypted Backup**
(msgs, contact, ...)

- Phone picks a secret AES key k
- ct = AES-GCM(k, data)
  ↳ sent to WhatsApp
- User saves key k (64 dec digits)

[There's a more complicated option that encrypts using a password.... uses hardware security device ... more complicated.]

Example: **Tahoe-LAFS**

Store file on remote server, indexed by hash of file.



Key k

h = Hash(ct)

Why hash? Server can check if ct got corrupted w/o client's help.

Key k, hash h

$h \stackrel{?}{=} \text{Hash}(ct)$

file = AES-CTR-Dec(k, ct)

Non-examples:

- Google Drive – no end-to-end enc by default – Google can see your data.
- Dropbox – can't use strong enc b/c of deduplication.

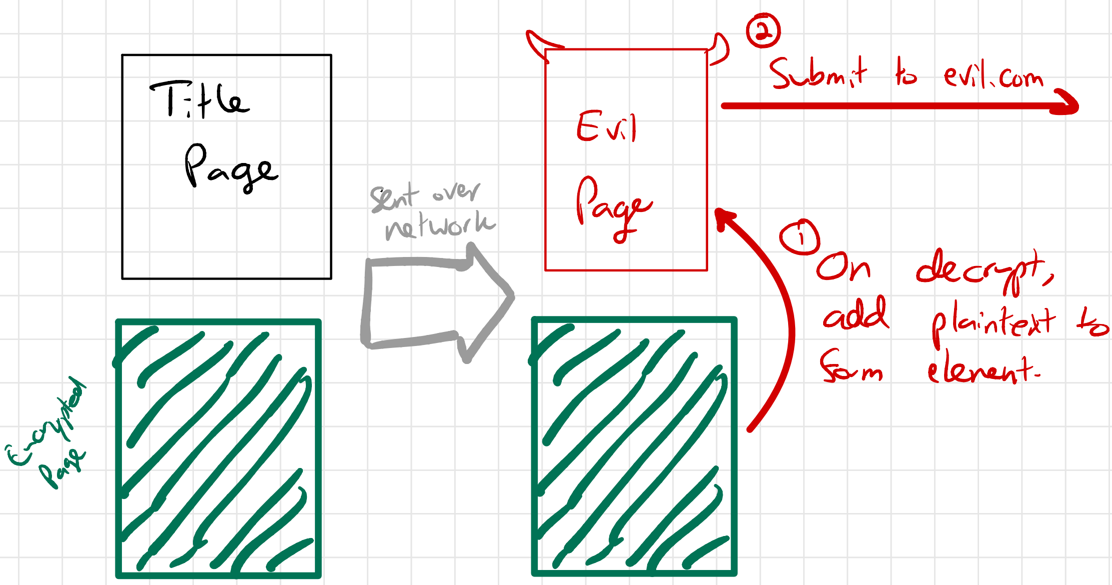
Even file encryption can be tricky...

(See pdf-insecurity.org)

<u>Example</u>: PDF v1.5

* PDF format allows password-encrypting some/all pages of doc — uses Hash(passwd) as AES key.
* PDF supports submitting form to external server via HTTP
* PDF forms can reference objects in doc
* PDF supports submit form on event (open, click, close)

→ Each seems fine on its own but together they allow an attacker to learn encrypted data.



Moral?

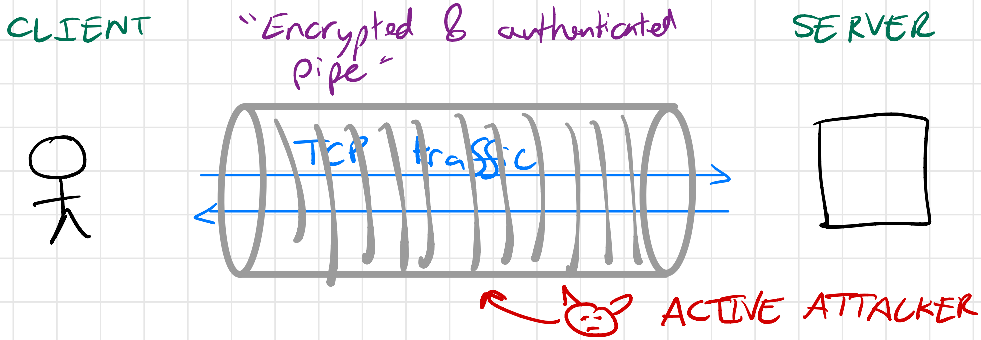
* As soon as you depart from the standard simple thing, you open the door to all sorts of subtle attacks...

* Auth'ing control info/metadata is as important as auth'ing the data itself.

What would have prevented this attack?
- MAC over entire PDF?
- Compatible w/ wanting to be able to load one page at a time?

Stream Encryption: TLS { Transport Layer Security (Formerly SSL)

Vision:



* Uses certificate-based pub key infrastructure to map domain name (mit.edu) → sig verif public key.

* Seems simple! Very hard to get right...
  Many attacks & patches since first versions.
  MORAL: Use TLS 1.3 — don't try it yourself.

Why is this hard?

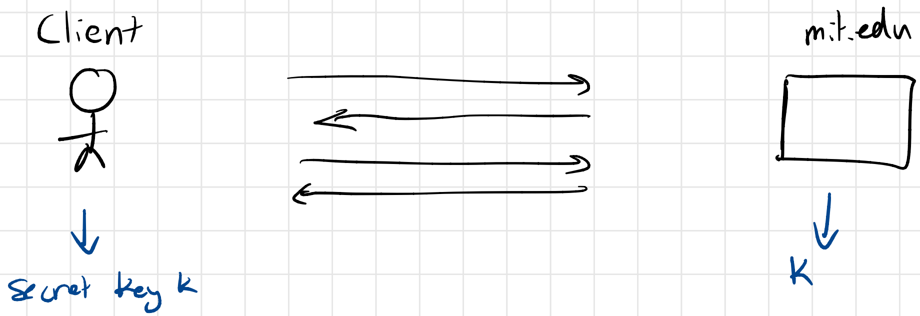
- Version/protocol negotiation — client and server may support different algs, protocols
  ↳ Downgrade attacks

- More complex protocol → more complex security goals.

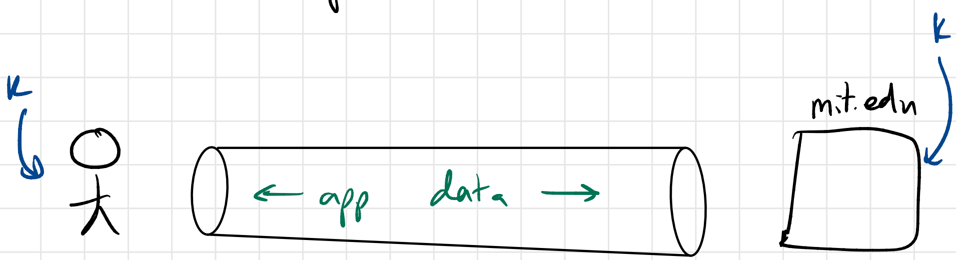
- FEATURES! Everyone wants to add something extra (e.g. client certificate auth at MIT)

Structure of TLS (v1.3)

I. Handshake (key exchange)



II. Record protocol



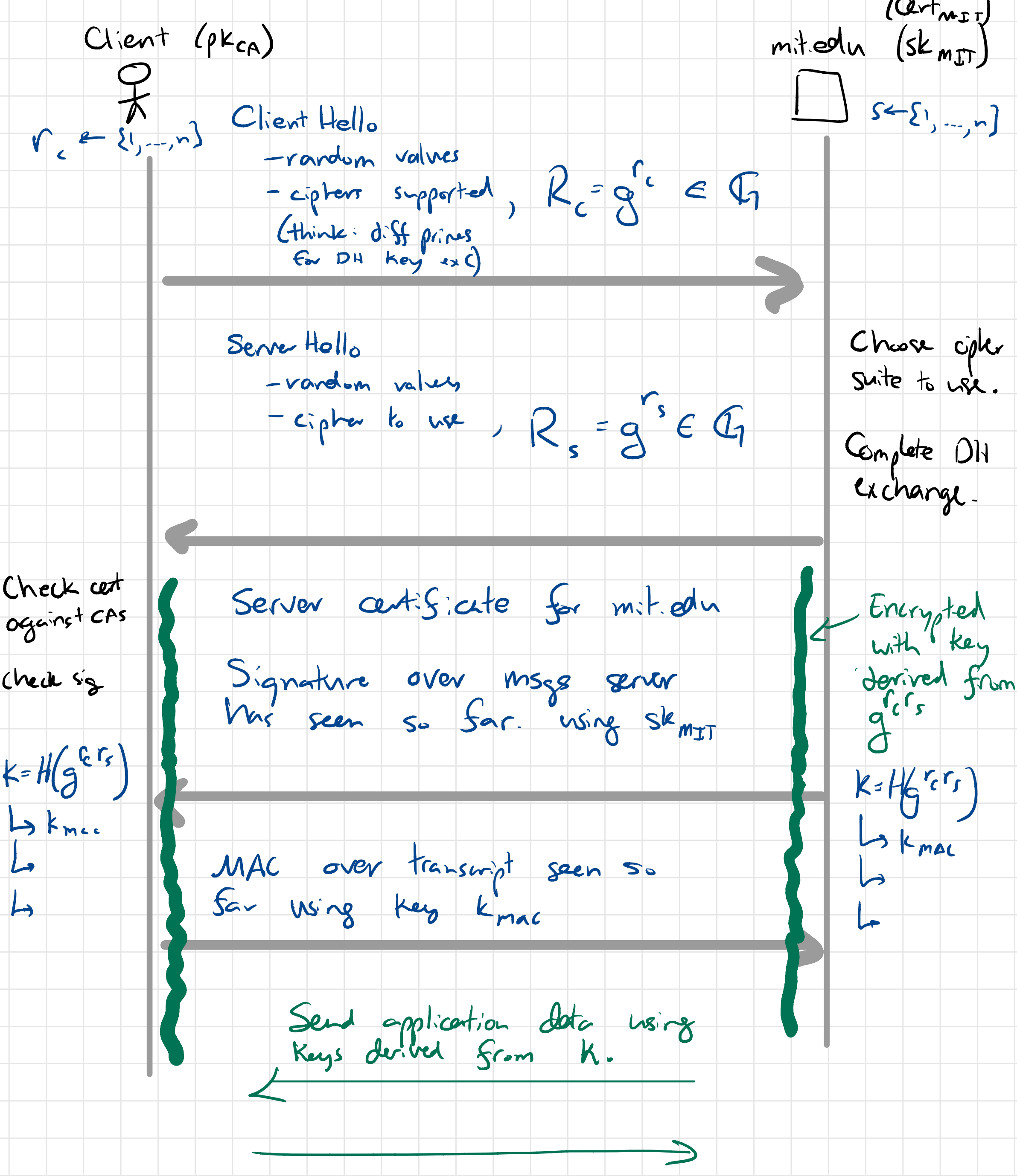
TLS Handshake: Properties

{ There are eight in the RFC!

* Correctness
* Security — adv "learns nothing" about session key → we saw this before
* Peer authentication — each party believes they're talking to the other
* Downgrade protection — parameters chosen should be the same if no attacker
* Forward secrecy w.r.t. key compromise
  ↳ If attacker compromises client/server, it cannot decrypt past traffic.
* Protection vs. key compromise impersonation
* Protection of endpoint identities

TLS Handshake

*Grossly simplified!



- Why replay attack isn't possible.
  - ↳ random values change every protocol run

- Why send server cert only after establishing shared DH secret?
  - ↳ Hides cert from passive network attacker (doesn't necessarily learn which Akamai-hosted site you're visiting)

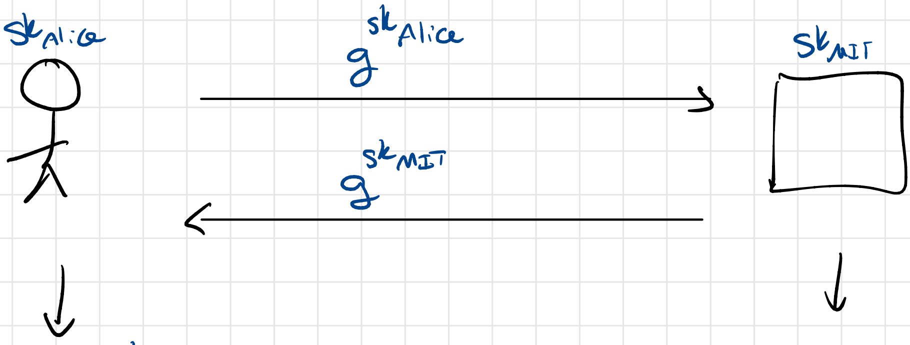
- Why does this provide forward secrecy?
  - ↳ Only use long-term secrets to sign
  - ↳ Delete the DH secret keys after handshake completes.

  [N.B. This doesn't protect past traffic against eavesdropper w/ big quantum comp.]

Key-Compromise Impersonation Attack

At MIT we use client certificates.

A bad way to do a handshake is this:



$k = H(g^{sk_{Alice} \, sk_{MIT}})$

$k = H(\; - - - \;)$

Problem: If attacker compromises Alice's secret key, attacker can pretend to be MIT to Alice.

- With sk_{Alice}, attacker can already make problems.
- But by impersonating MIT attacker can trick Alice into sending more data. (passwd, etc.)

Properties that TLS doesn't provide

Authenticated EOF

- TLS makes data available to app as it arrives
- Needed for many uses (Youtube, etc.)
- But counterintuitive consequences:

```
curl https://sh.rustup.rs | sh
```

↳ rm -rf /tmp/install....

✂ CUT!

... what is the right thing to do here?

Hiding length of plaintext ("CRIME")

Reasonable thing to do: gzip data before sending it to TLS (used to be standard).

Problem: Attacker controlled data often sent in same stream as secrets. Esp in web

```
GET /a HTTP/1.1
Cookie: <secret>
```
} 123 bytes

```
GET /b HTTP/1.1
Cookie: <secret>
```
} 122 bytes

!

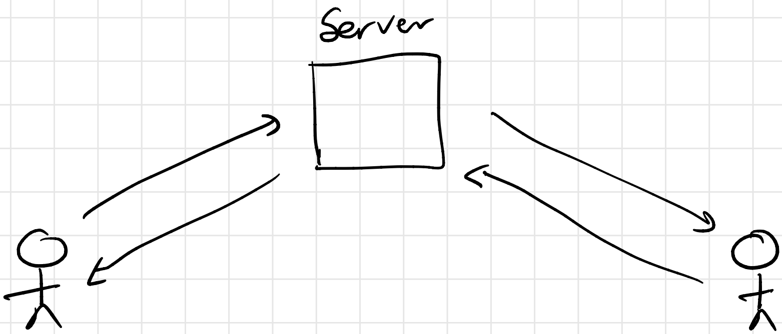
↳ Compression now deprecated in TLS.

Moral: Use TLS 1.3 whenever you need "encrypted TLS"

Be aware of its pitfalls.

Encrypted Messaging

Think: Signal, WhatsApp, iMessage, ...



Why different from stream setting?

* "Connections" are long lived – for years
* Little data, few connections
* non-interactive – either party can be offline for long periods

Goals: Many as in TLS (though underspecified)

e.g. Forward Secrecy

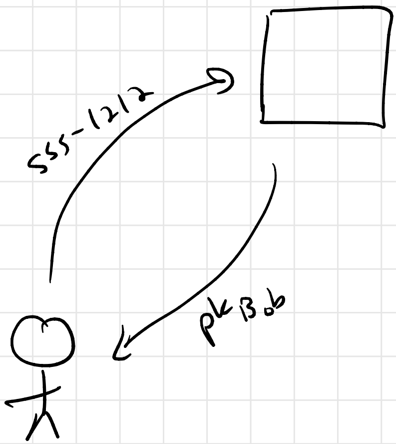
"Post-Compromise Security" – If attacker gets a snapshot of your device, will eventually not be able to read msg.

Not clear how relates to real-world threat

Unlike TLS, these apps typically rely on a centralized key server.

↳ to map [phone #] ⟹ [Public key]

If someone compromises the key server, very weak protection against active attack.

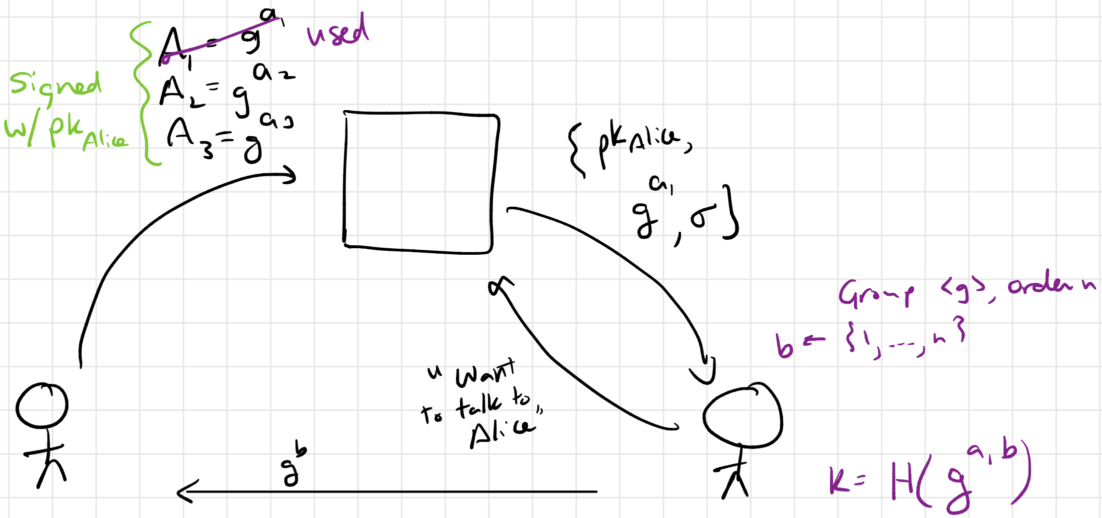


* Phone can show you hash of claimed pk Bob ... check manually. "No one" does this

* App can give warning when pk Bob changes
↳ "Everyone" ignores this

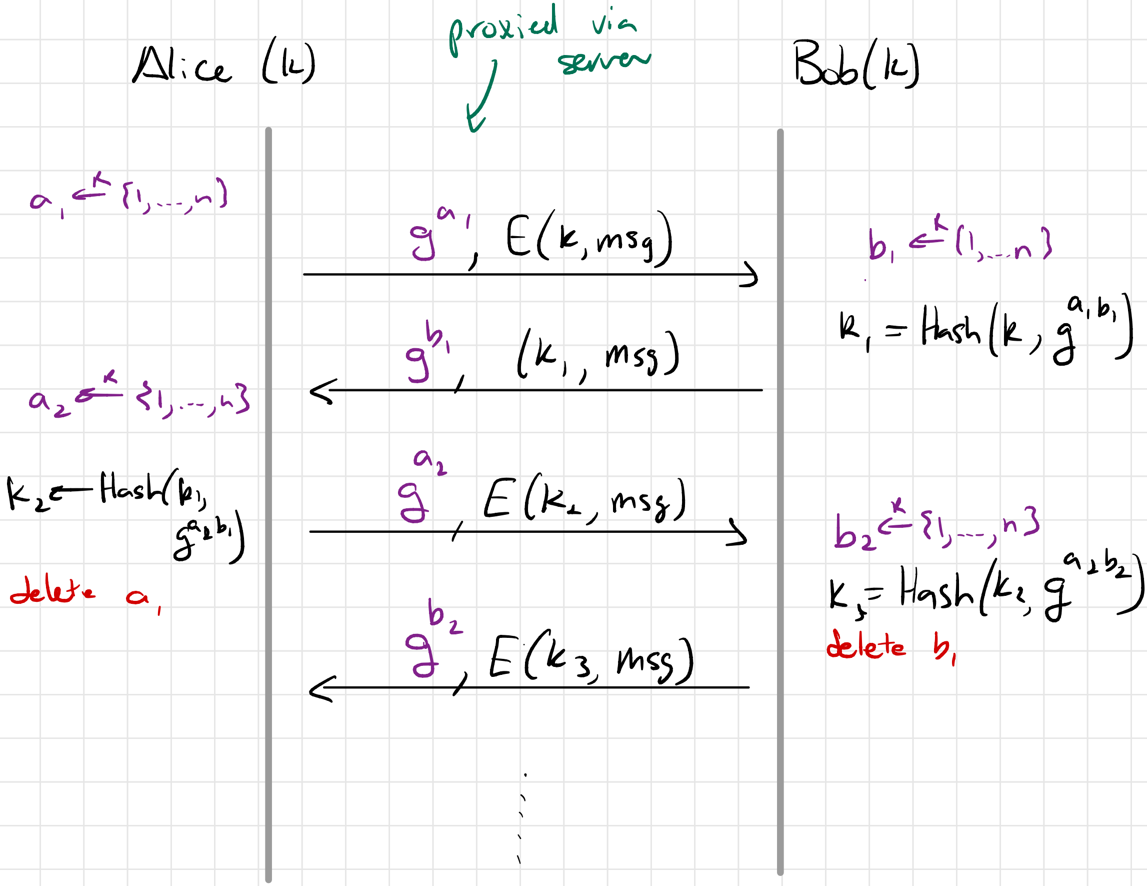
↑ For sec-conscious users, maybe these suffice?

Toy Key Exchange



N.B. Server learns who is talking to whom.

Toy Ratchet – How to get forward secrecy and post-compromise security.



- An attacker who compromises device cannot recover past msgs
- Without persistent compromise, protocol will "heal" security

* Big advances in encrypted comms in last ~10 yrs
    - ↳ Before that: not much TLS, not much enc msging
* Next time: Open problems... what we haven't solved.

The End!