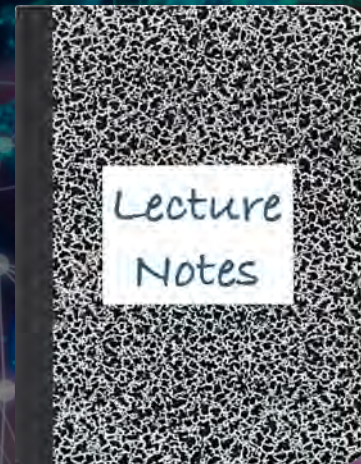


CS 419 – Computer Security

Week 4

Authentication and Secure Communication

Paul Krzyzanowski



© 2026 Paul Krzyzanowski. No part of this content may be reproduced or reposted in whole or in part in any manner without the permission of the copyright owner.

What We Will Cover

Secure communication

Getting a shared key, the adversary on the network, freshness, and challenge-response

Key distribution

Needham-Schroeder, its flaw, and Kerberos

Public keys and TLS

Signatures, signed Diffie-Hellman, forward secrecy, and the TLS handshake

Passwords

Sending, storing, guessing, and defending them

Beyond passwords

One-time codes, second factors, real-time phishing, and passkeys

A Replayed Car Key (2015)

1 First press

RollJam jams the car and records code 1.

The car does not respond.

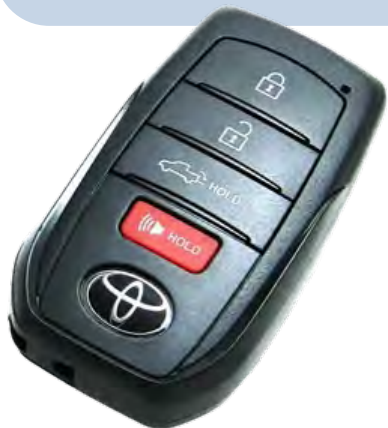
2 Second press

RollJam jams and records code 2, then sends code 1.

The car unlocks.

3 Later

The attacker sends code 2. The car has never seen it and accepts it.



Each press sends a new code from a counter shared with the car, and the car accepts only codes it has not seen.

Nothing was decrypted and no key was stolen.

The replayed code was authentic.

Secure Communication

From Primitives to Protocols

A cryptographic primitive

One operation: hash, MAC, signature, cipher.

It protects a single message.



A security protocol

A sequence of messages exchanged to reach a goal, such as proving an identity or agreeing on a key.

An adversary can:

replay, ***reorder***, and ***mix*** messages from different runs

A protocol can fail while every cipher, MAC, and signature in it works correctly.

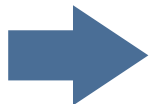
Key Explosion Problem: Getting a Shared Key

Every communication channel needs a secret key

Alice ↔ Bob
Alice ↔ Carol
Alice ↔ Dave
Bob ↔ Carol
Bob ↔ Dave
Carol ↔ Dave

1,000

employees



499,500

keys for every pair to
communicate

$n(n - 1) \div 2$
keys for n parties

Pairwise keys do not scale

Each new hire adds a key for every existing employee.

Changing one key means reaching both holders again.

Strangers cannot start

A browser visiting a site for the first time does not share a key with it. Something has to create one.

Four Ways to Get a Shared Key

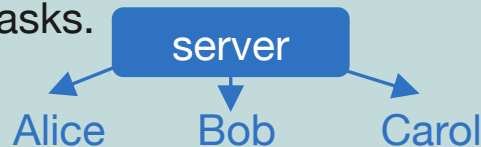
Pre-shared key

Distributed out of band, like a Wi-Fi password printed on a card.

Alice →  → Bob

Trusted third party

A server shares a long-term key with every participant and issues keys to any pair that asks.



Public key transport

One party generates a key and encrypts it with the other party's public key.

$E_{\text{bob_public_key}}(K) \rightarrow \text{Bob}$

Diffie-Hellman

Two parties compute a shared secret from public values. It must be authenticated.

Alice_public_value → Bob
Alice ← Bobs_public_value → shared secret

Long-Term Keys and Session Keys

Long-term key

Identifies a party for months or years.

Examples: a password-derived key, a server's private key.

Session key

Protects one conversation and is erased when it ends.

Why protocols create session keys:

A leaked session key
exposes one
conversation

Less data is encrypted
under any one key

It can be erased when
the conversation ends

The Threat Model: What the Adversary Can Do

Read and record

every message

Delete, delay, reorder

messages in transit

Replay

old messages,
in this run or a later one

Inject

messages of its own

Start communication

with any party, or several
at once

Participate

as a legitimate user with
its own keys

The attacker cannot break the cryptography and does not have the keys

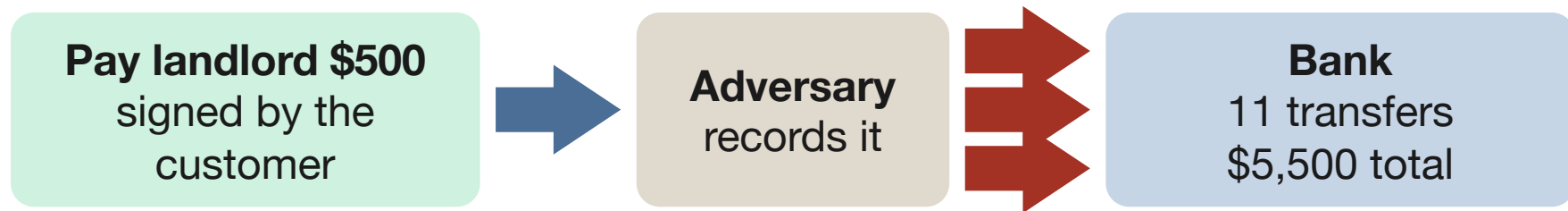
The Adversary in the Middle (AITM)



The adversary reads anything that is not encrypted and can change anything that is not authenticated.

The position can come from a compromised router, a rogue Wi-Fi access point, a poisoned DNS entry, or a phishing link.

Replay Attack: Replaying a Valid Message



Every copy passes signature verification.
Nothing in a MAC or a signature says when a message was produced.

A **replay attack** reuses a valid message outside the context in which it was sent.
The defense is to let the receiver tell a current message from an old one.

Ensuring Freshness of Messages

Mechanism	How it works	Cost
Nonce	Alice sends an unpredictable value and accepts only a reply that contains it	A round trip. Protects only the party that chose it.
Timestamp	The sender includes the time, and the receiver accepts only recent messages	Synchronized clocks. A replay within the window succeeds.
Sequence number	Each message has an increasing counter	Both sides keep state and handle gaps and restarts.

A message is **fresh** if the receiver has evidence that it belongs to the current exchange or an acceptable recent period.

Nonce: Two Meanings

Nonce = “*number used once*” – *randomly generated each time*

Nonce inside an algorithm

AES-GCM needs a different nonce for every encryption under a key. ECDSA needs a secret per-signature value.

Reusing one can expose messages or the signing key.

Nonce for freshness

A value one party chooses so that it can recognize a reply as belonging to this exchange.

It can be sent openly. It must be unpredictable.

Same word, same rule against repeating, and different jobs.

Basic idea of freshness:



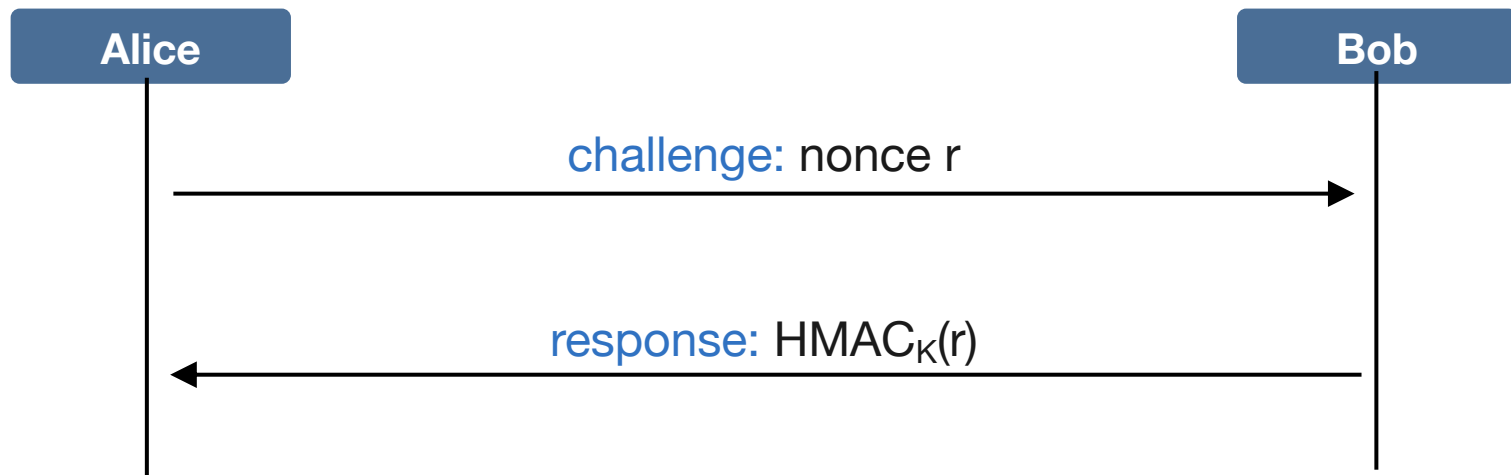
“Bob, prove to me that you can encrypt this string: 0x706f6f7079”

Security Protocol Notation

Notation	Meaning
A, B, T	Alice, Bob, and Trent, the trusted third party
$A \rightarrow B: X$	Alice sends message X to Bob
K_A, K_B	Long-term keys shared with Trent
K_{AB}	A session key for Alice and Bob
$\{M\}_K$	M encrypted with K, tampering detected
r_1, t	A nonce (a random string) and a timestamp
$X \parallel Y$ or X, Y	Fields sent together in one message

The notation shows the messages, not the checks each recipient performs.

Challenge and Response



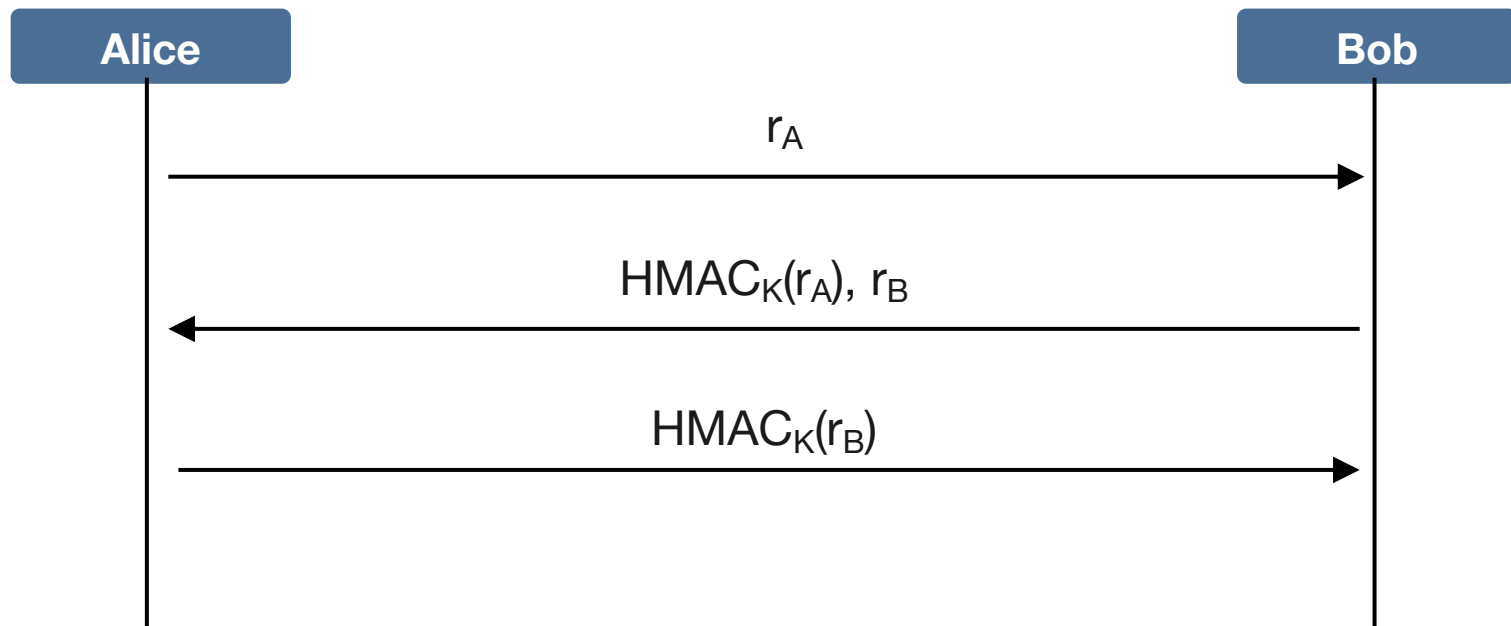
Only a holder of K can compute the response.

The key never crosses the network.

A recorded response will not match a new challenge.

A live one can still be relayed.

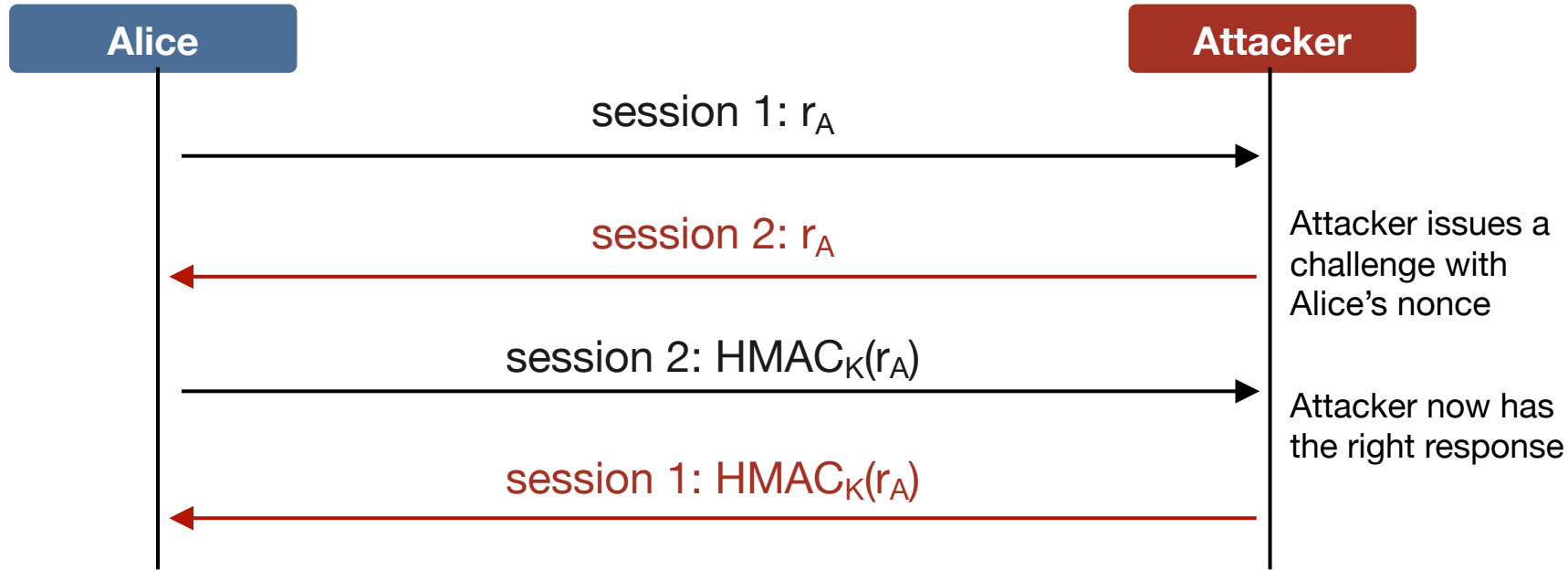
Mutual Authentication



In **one-way authentication**, Alice authenticates Bob.

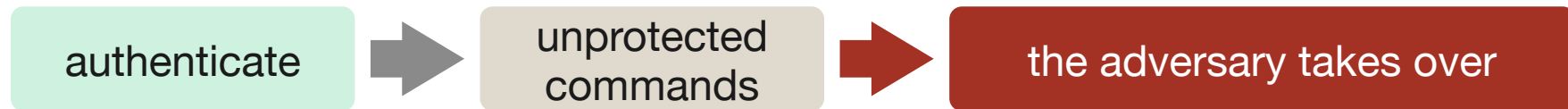
In **mutual authentication**, each party authenticates the other.

The Reflection Attack

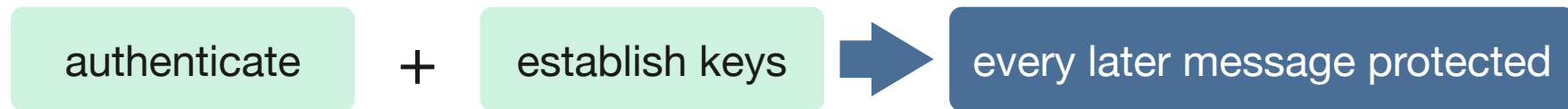


Fix: make the directions distinguishable, by adding an identity, like $\text{HMAC}_K(\text{"Bob"}, r_A)$, or use a different key in each direction.

Protecting the Conversation



This is **session hijacking**.



This is an **authenticated key exchange**.

Authentication can also take place inside a channel that is already protected, as with a password typed into a web page served over TLS.

Key Distribution Centers

A Key Distribution Center (KDC)

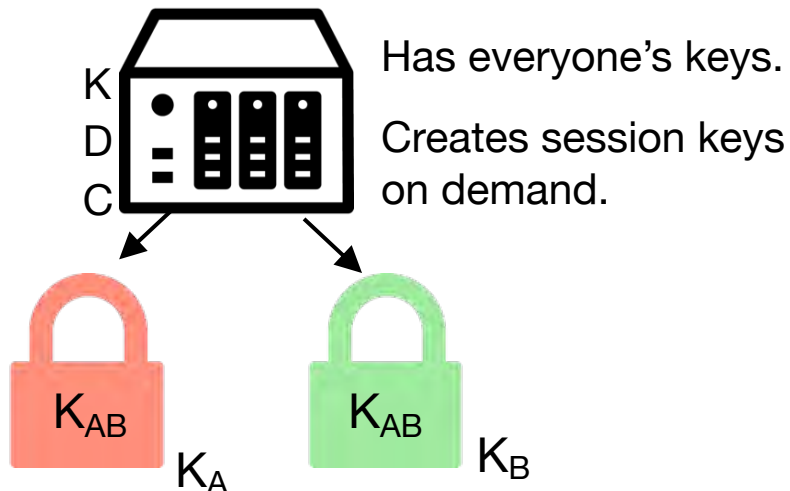
Each participant shares one long-term key with a trusted server.

The server creates a **session key** for any pair and sends each party a copy encrypted under that party's key.

Avoids the problem of each user or service having to manage a secret key for every service they might need to access

The KDC can also manage **authorization**.

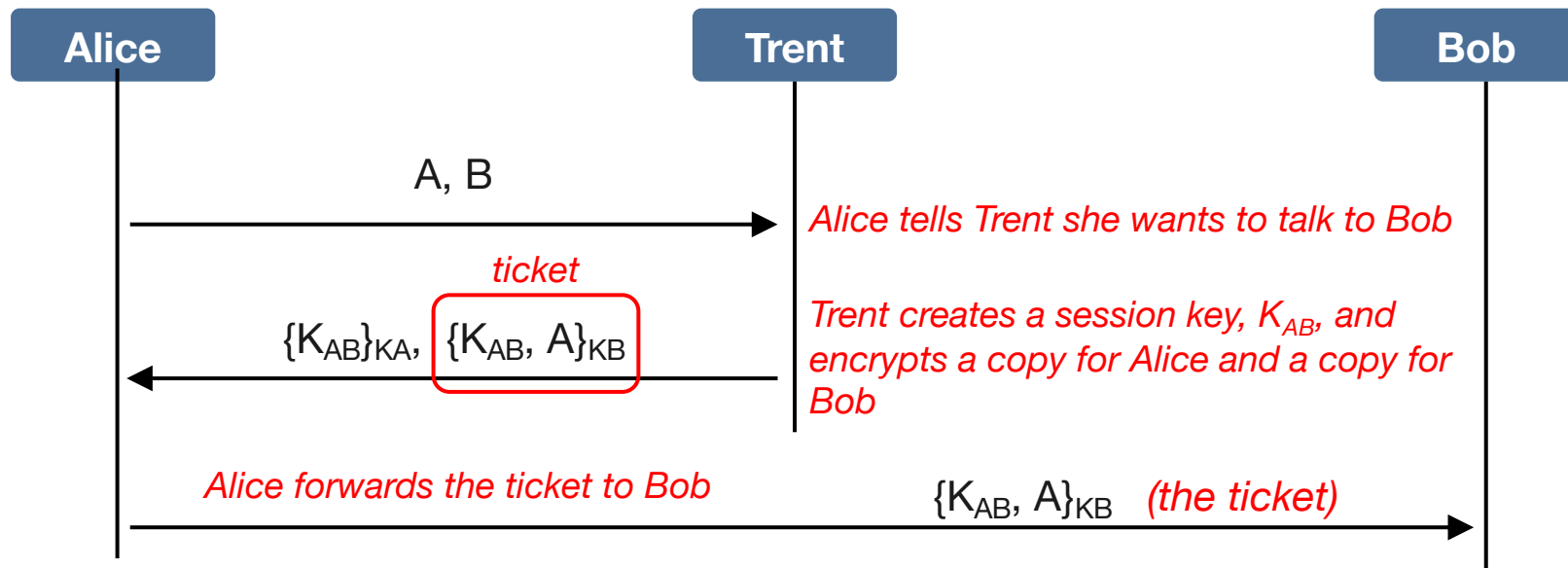
If Alice isn't allowed to talk to Bob, she won't get a session key.



The risk

All the trust sits in one place.
The server knows every key.

A Basic Key Distribution Protocol



Alice cannot read Bob's ticket.
Bob learns that Trent issued the key for Alice, but **not when**.

Replaying an Old Ticket



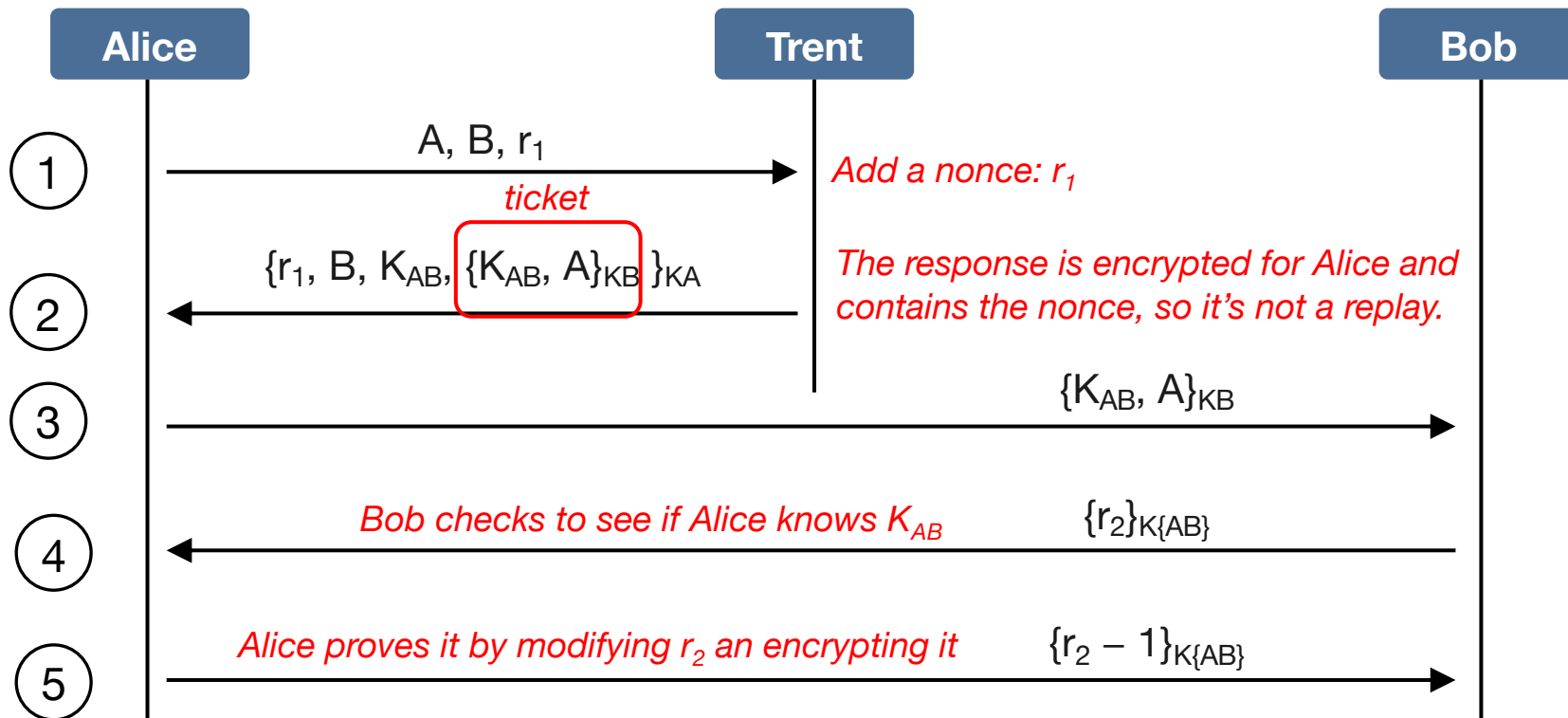
If the old key leaked

From a memory dump or a careless log file, the adversary can now impersonate Alice.

Even if it did not

The adversary can push Bob back to using a key Alice has discarded

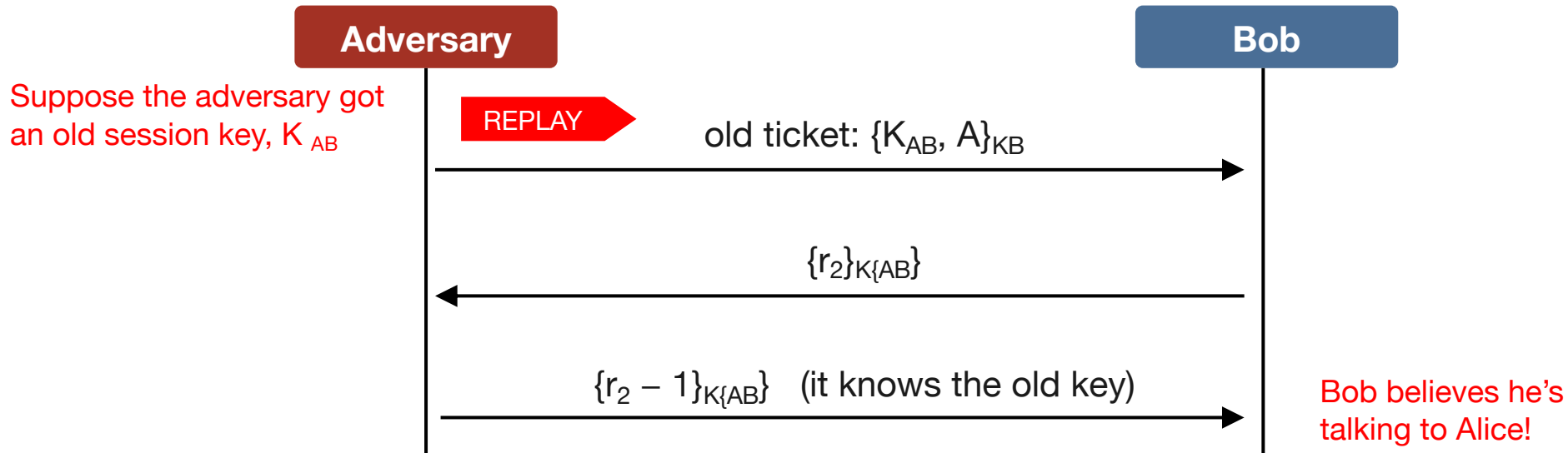
The Needham-Schroeder Protocol



Why Each Field Is There

Field	Why it is there
r_1 in messages 1 and 2	Alice knows Trent's reply answers this request. An old reply will have an old nonce.
B in message 2	An adversary cannot redirect Alice's request to someone else.
Ticket inside message 2	Sealed under Bob's key. Alice forwards it and cannot alter it.
r_2 in messages 4 and 5	Bob learns that someone holding K_{AB} is responding now.
$r_2 - 1$	Alice's reply differs from Bob's message, so it cannot be sent back to him.

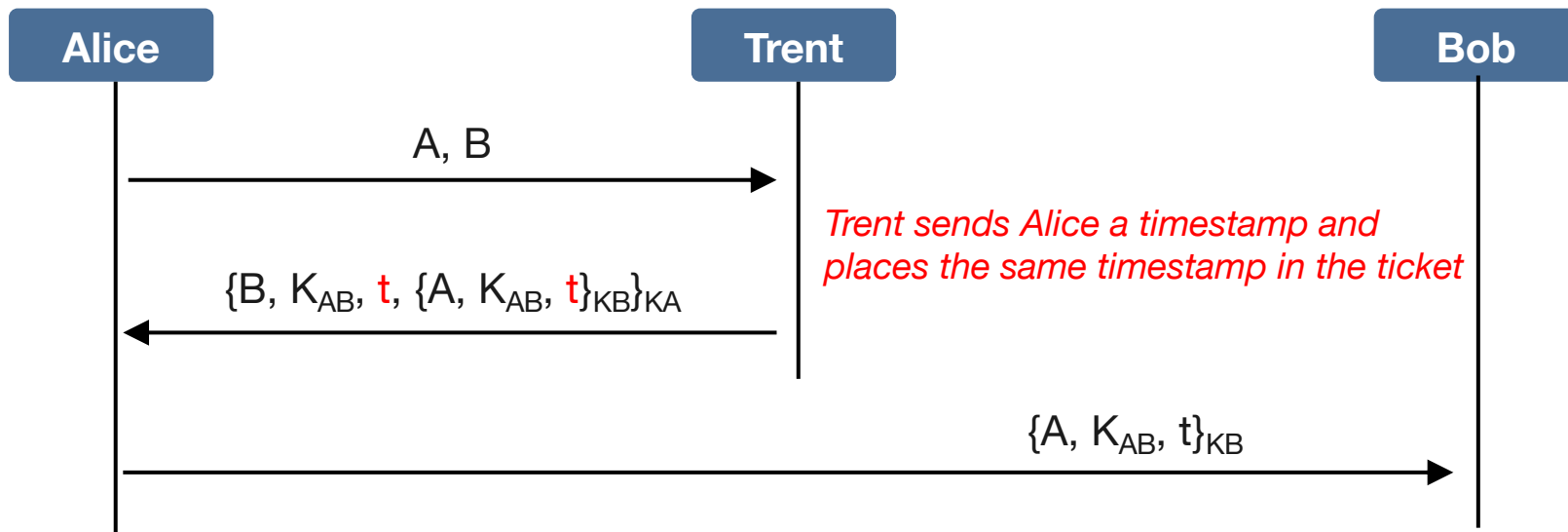
A Flaw Found Three Years Later



Alice's nonce shows **Alice** that Trent's reply is fresh.

Bob's nonce shows **Bob** a fresh response, not a **fresh key**.

Denning-Sacco Fix: Timestamps in the Ticket



What t gives Bob

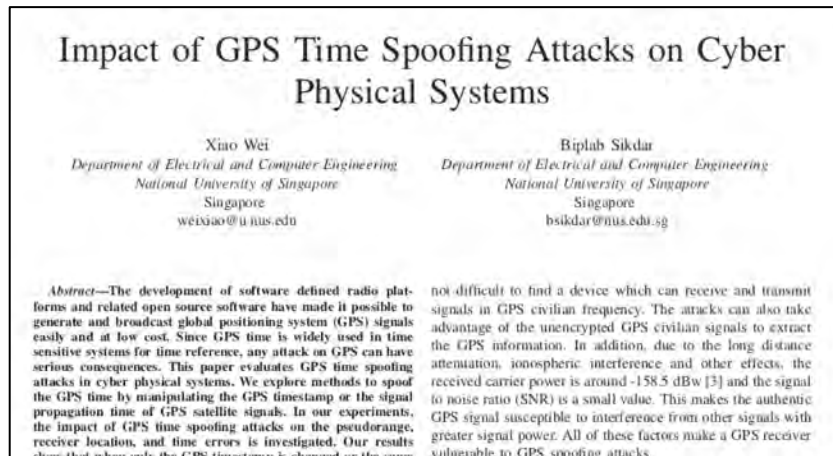
His own evidence that Trent issued the key recently

What it costs

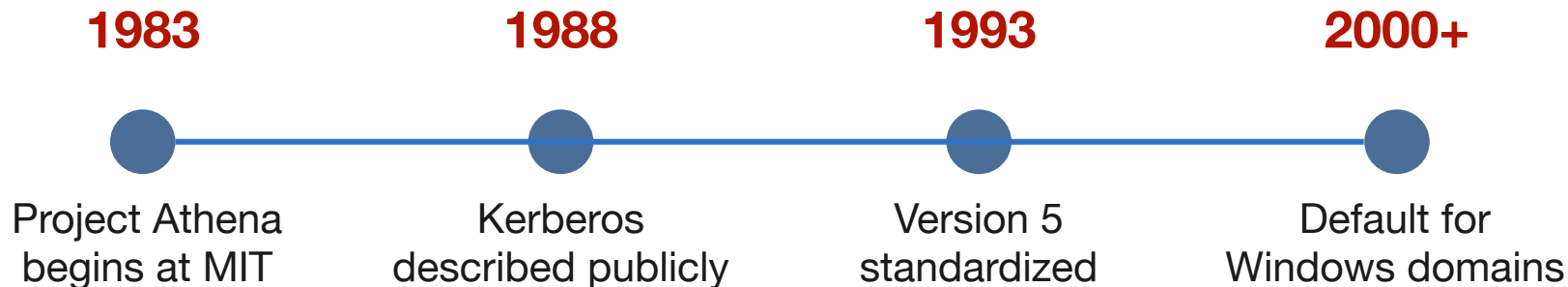
Synchronized clocks, and a window in which replays still work

The Risk of Timestamps

- Use of timestamps relies on synchronized clocks
 - Messages may be falsely accepted or falsely rejected because of bad time
- Time synchronization becomes an attack vector
 - Create fake NTP responses
 - Generate fake GPS signals



Kerberos



The problem

Thousands of users, many services.

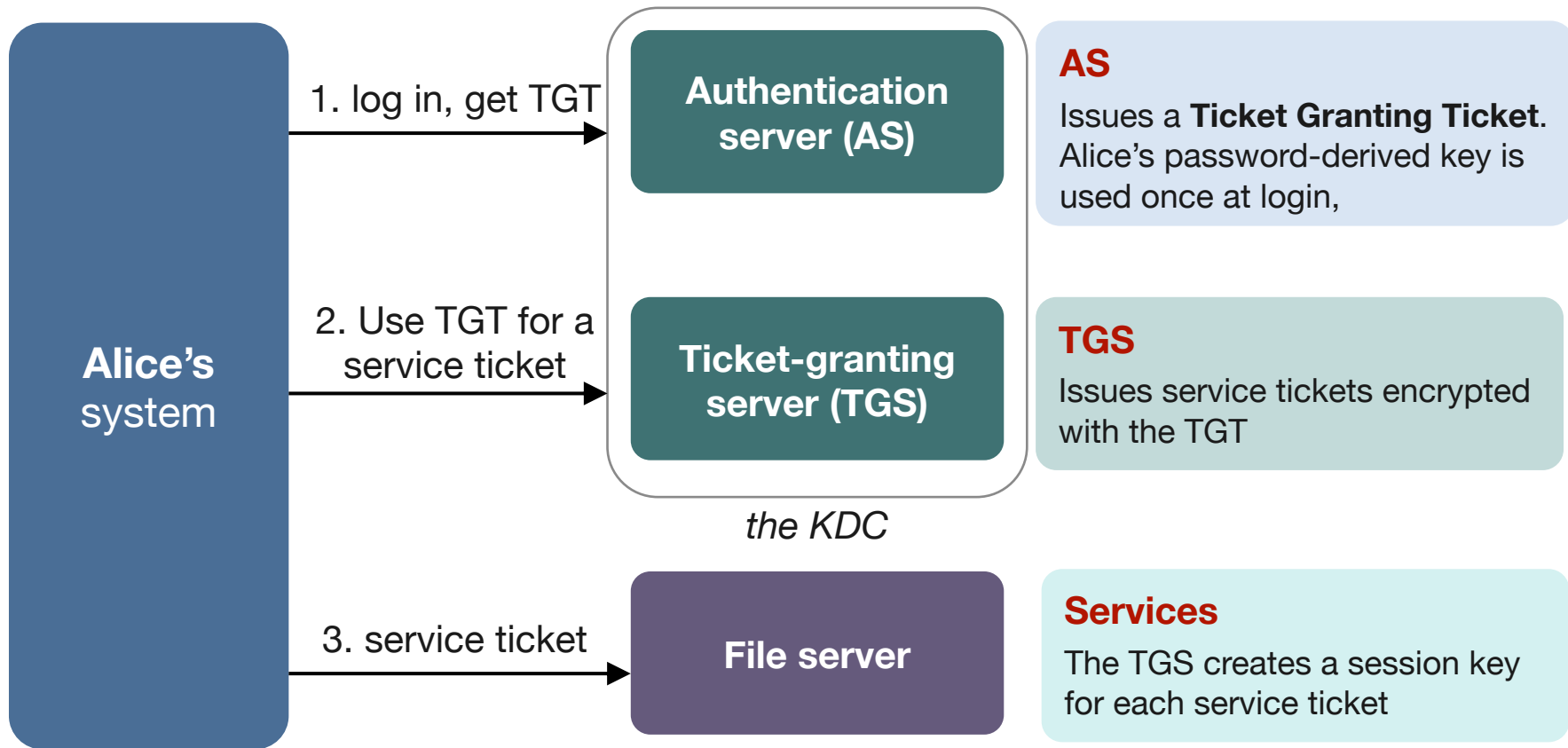
Trusted machines, untrusted networks.

Each user should log in once, and passwords should never cross the network.

The approach

Needham-Schroeder with a key distribution center, plus the Denning-Sacco timestamps

The KDC is Split Into Two Servers



Tickets and Authenticators

Ticket

The client's name and a session key, encrypted under the service's key.
It can be copied.

+

Authenticator

The client's name and the current time, encrypted under the ticket's session key



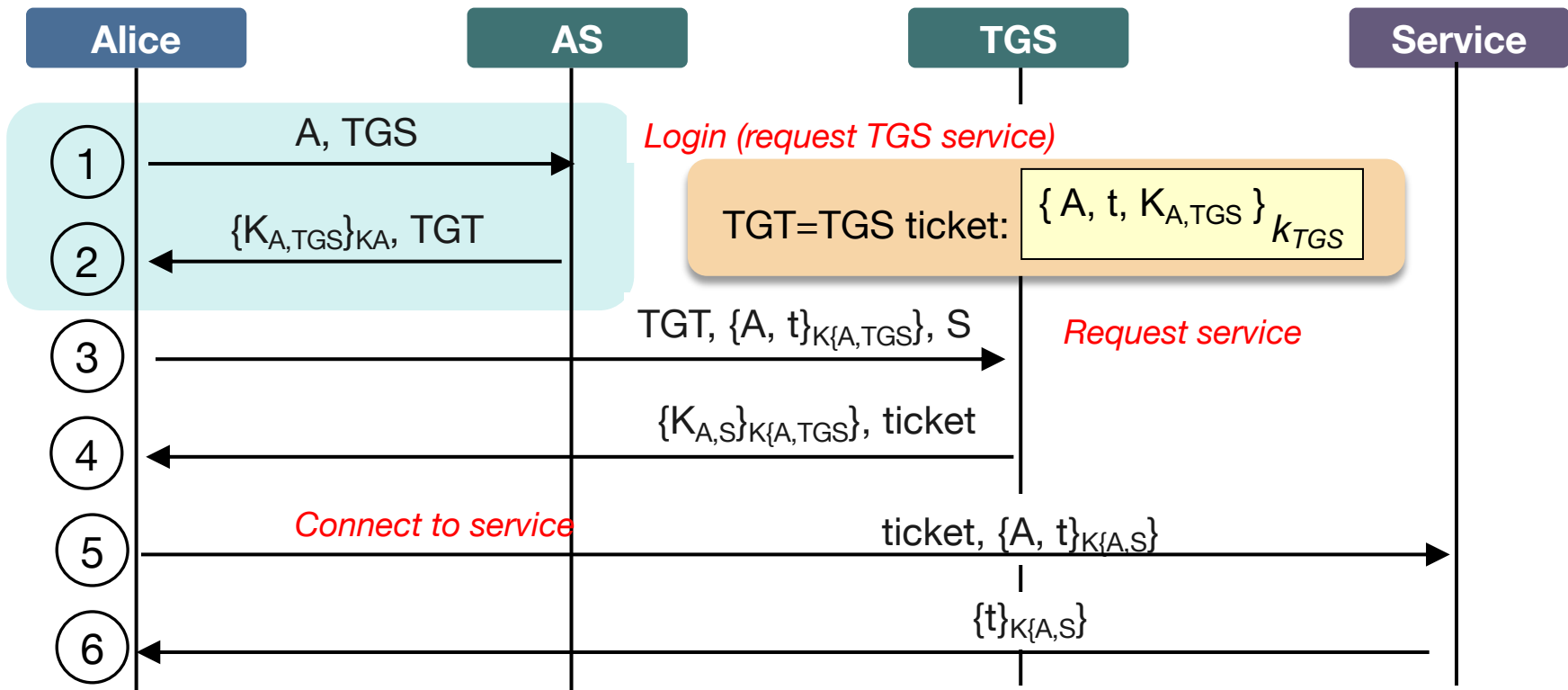
Proves

The presenter holds the session key, not just a copied ticket

Freshness comes from the Denning-Sacco fix:

Tickets have validity periods, **authenticators have timestamps**, and services reject authenticators more than five minutes off and remember recent ones.

The Kerberos Exchange



Messages 1-2: once per login. 3-4: once per service. 5-6: each connection.

The Cost of a Central Key Server

Availability

The KDC must be up to issue new tickets.

Organizations run replicas to ensure uptime.

A single target to attack

The KDC holds the long-term keys of every user and service.

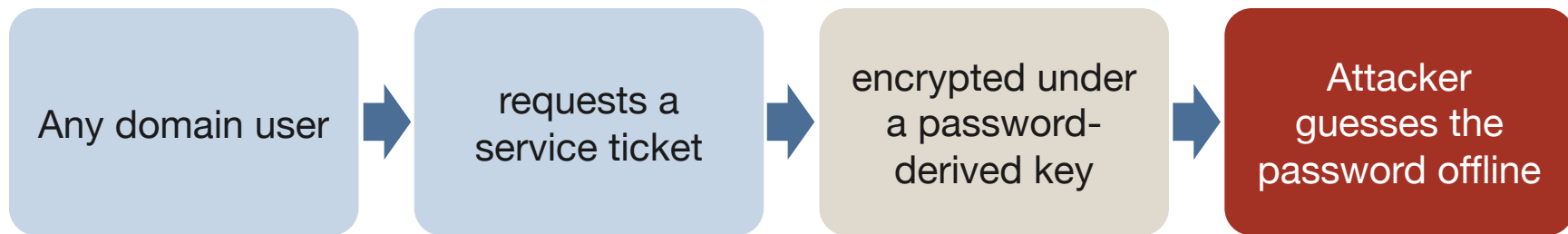
Compromising it compromises everything.

Synchronized clocks

Every machine needs a clock synced within the tolerance of the KDC's, or authentication fails.

Each cost is due to the design, not a bug.

Guessing Passwords From Tickets



A key derived from a password is only as strong as the password.
Guesses do not produce failed logins: you just get a ticket you can't decrypt.

Defense

Long, randomly generated passwords for service accounts, or accounts whose passwords are managed automatically

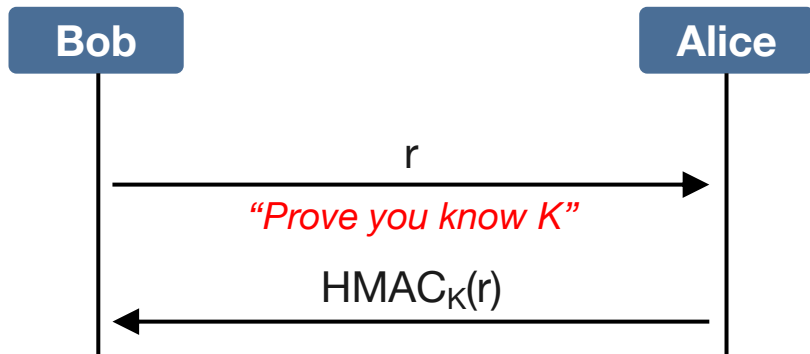
What Each Protocol Added

Protocol	What it added	Remaining limitation
Basic protocol	A ticket Alice forwards and cannot read	Nothing shows the key is new
Needham-Schroeder	Nonces and proof of session-key possession	Bob cannot tell whether the key is recent
Denning-Sacco	A timestamp inside the ticket	Clocks, and a separate proof of key possession
Kerberos	AS and TGS , single sign-on, authenticators	The KDC, clocks, and strong passwords

Public Key Authentication and TLS

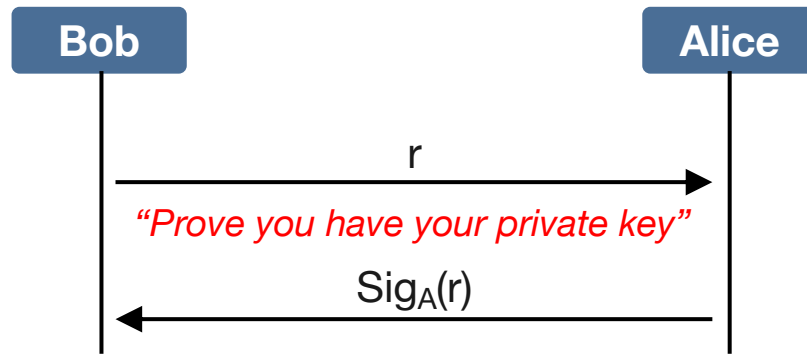
Authenticating With a MAC or a Signature

With a shared key K



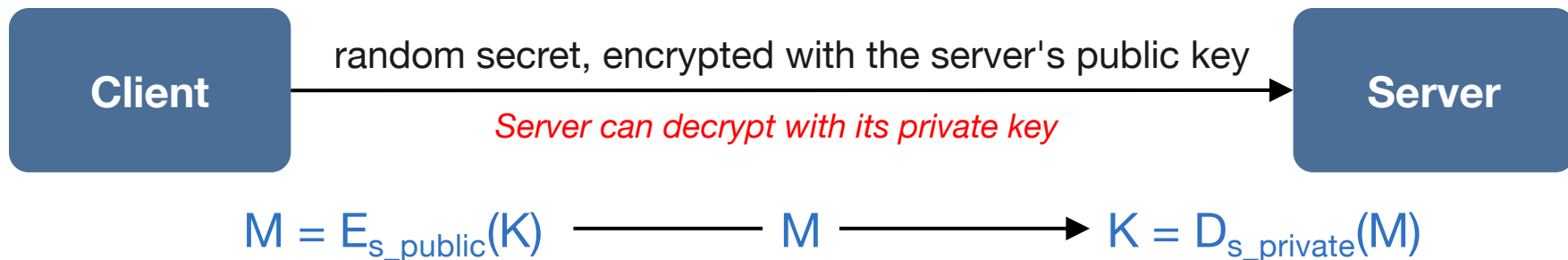
Bob holds K , the same key Alice uses.
Anyone who steals it from Bob can pretend to be Alice.

With a public-private key pair



Bob holds only Alice's public key from her certificate and can verify Alice's signatures
Nothing he stores lets an attacker sign as Alice.

Sending a Key Under a Public Key



Both sides derive session keys from the secret.

TLS through version 1.2 offered this with RSA.

The weakness

The server's **long-term private key** decrypts every secret it ever received.

Anyone who records traffic today and obtains the key later can read all of it.

Example: Lavabit's Private Key (2013)

Lavabit used RSA key transport:

Each client generated a random secret & encrypted it with Lavabit's public key.

1

user under investigation

≈ 400,000

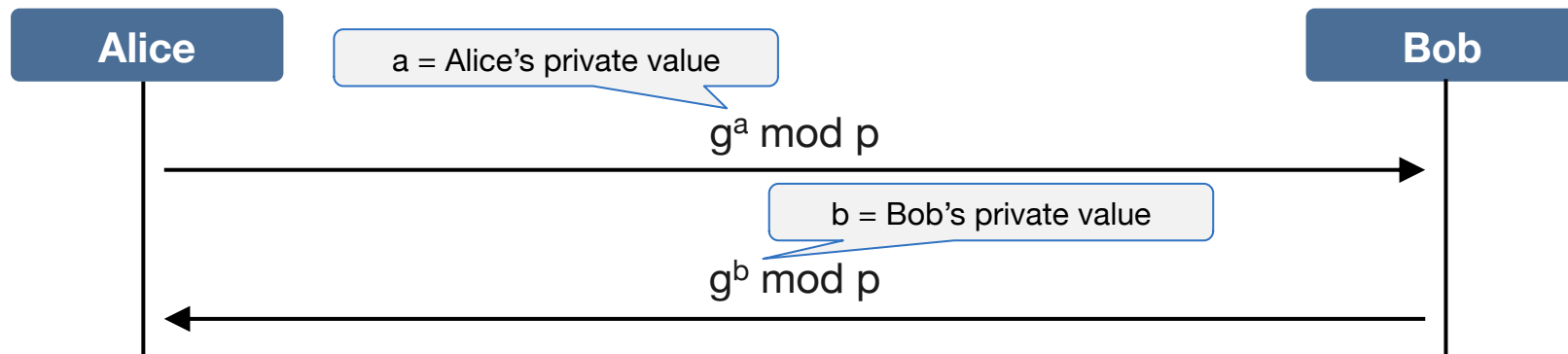
Lavabit users at risk

In 2013, the FBI demanded the **TLS private key of Lavabit**, an encrypted email service whose users included Edward Snowden.

With RSA key transport, the key could expose every user's recorded connections.

Ordered to provide a usable digital copy, the owner complied and shut the service down on August 8, 2013.

Diffie-Hellman: Review



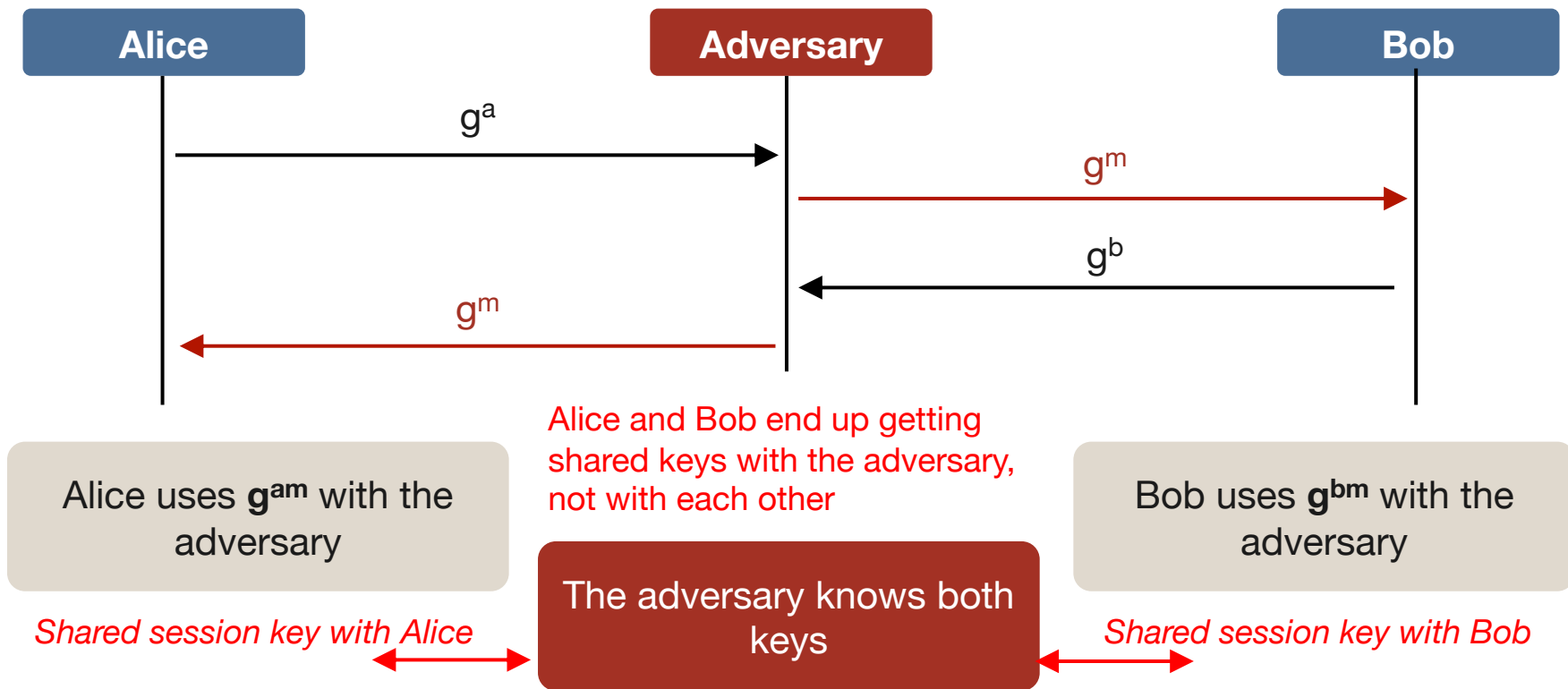
Alice computes $(g^b)^a$

Bob computes $(g^a)^b$

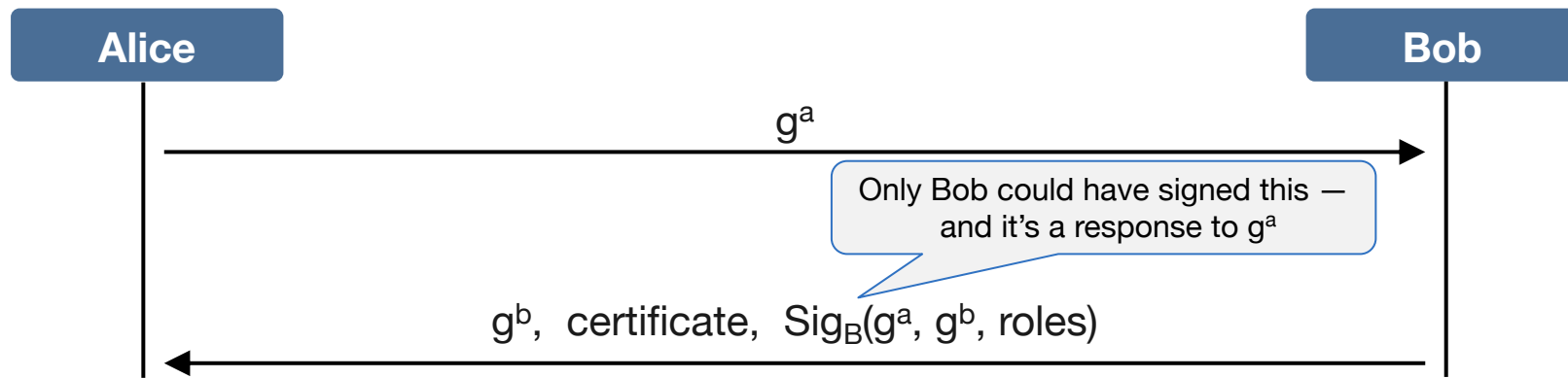
both get g^{ab}

Diffie-Hellman does not authenticate anyone.
Nothing in g^a says who sent it.

An Adversary in the Middle of Diffie-Hellman



Signing the Exchange



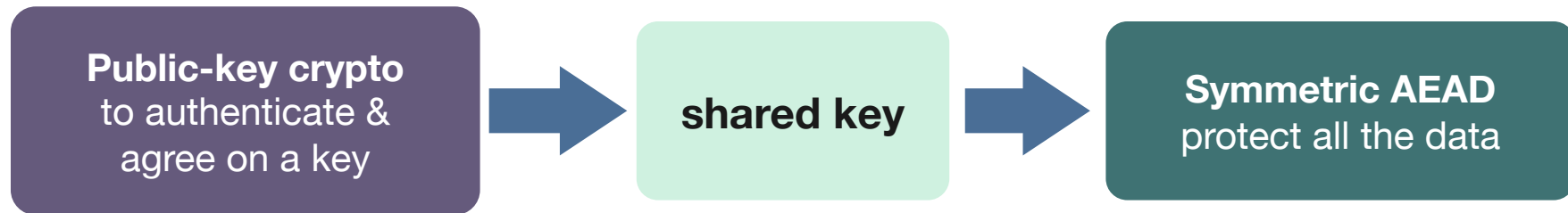
Alice verifies Bob's signature with the public key in his certificate.

A substituted g^m does not carry a valid signature from Bob.

Bob's **long-term key** only **signs**.

The shared secret comes from the Diffie-Hellman values, not from decrypting anything.

Hybrid Cryptosystems



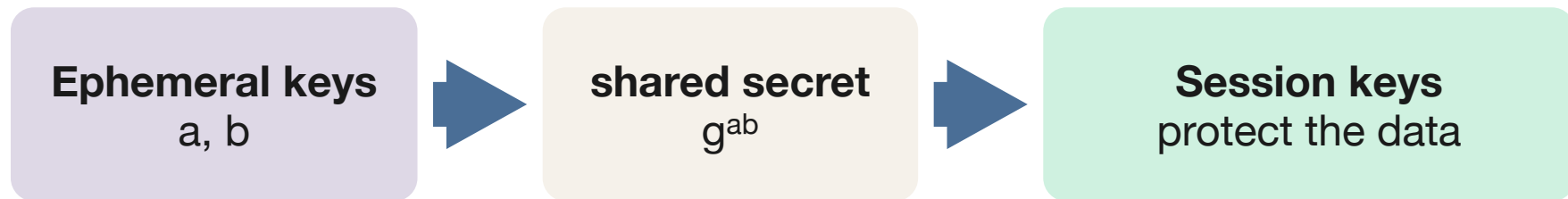
This combination is a **hybrid cryptosystem**.

TLS and PGP email encryption both use it.

Not to be confused with hybrid key establishment

Hybrid key establishment combines a classical key exchange with a post-quantum one to create a shared secret, so the key stays secret if at least one algorithm remains secure. A hybrid cryptosystem can use hybrid key establishment.

Long-Term, Ephemeral, and Session Keys



Long-term key: signs the exchange, and lasts across many connections
Examples: password derived keys, public-private keys with X.509 certificates

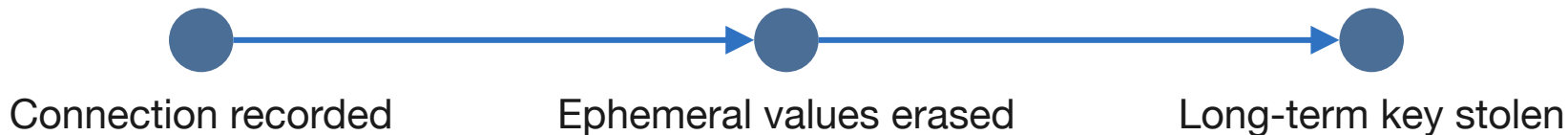
Ephemeral keys

Used only to compute the shared secret, then erased. They never encrypt data.

Session keys

Derived from the shared secret. They protect the rest of the connection.

Forward Secrecy



RSA key transport

The stolen key decrypts the recorded secret. Every past session is exposed.

Ephemeral Diffie-Hellman

The recording holds only g^a and g^b . The key never touched the secret.

Forward secrecy: compromising a long-term key does not expose sessions recorded before the compromise.

Why SSL Was Created

1994: web traffic on the Internet was unencrypted. A credit card number typed into a form could be read or changed on the way.

Application (HTTP, email, ...)

SSL / TLS

TCP

IP

The application reads and writes a byte stream, just like it would with a socket

Secure Sockets Layer

Netscape's answer.

- SSL 2.0 shipped in 1995
- SSL 3.0 in 1996.
- HTTP over SSL became HTTPS on port 443

Transport Layer Security (TLS)

IETF published the protocol as TLS in 1999

- Current version is TLS 1.3, from 2018

What TLS Has to Do

1

Agree on algorithms

2

Authenticate the server

3

**Establish a fresh
shared ephemeral key**

4

**Protect every record
that follows**

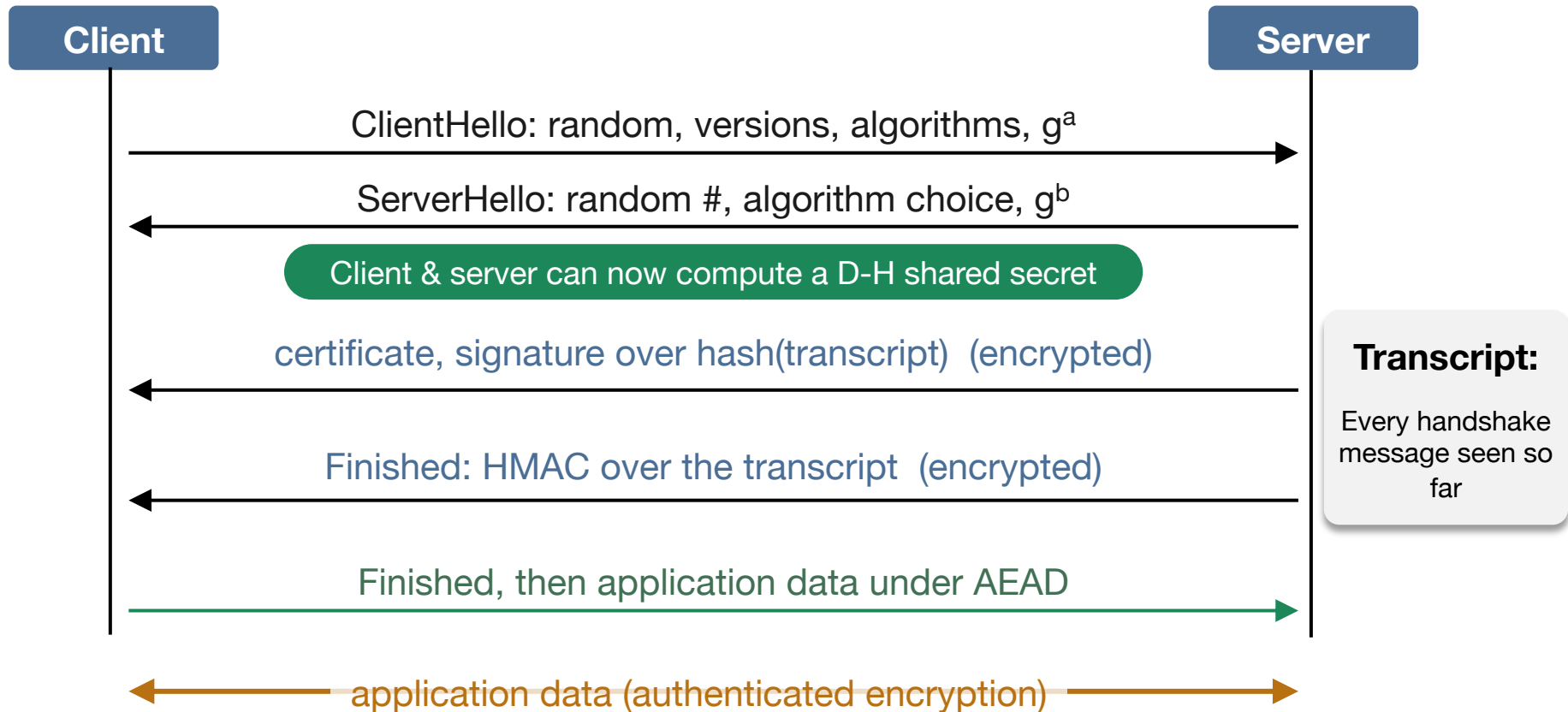
The **handshake** does the first three.
The data stream is then sent in protected chunks called **records**.

TLS Handshake



App Communication

The TLS 1.3 Handshake



The Role of Each Piece in a TLS Session

Component	Job
Ephemeral Diffie-Hellman values	A fresh shared secret with forward secrecy
Certificate chain	Binds the server's public key to its domain name
Signature over the transcript	Proves the server holds the private key, for this connection
Finished HMACs	Both sides computed the same keys and saw the same messages
AEAD-protected records	Encryption and integrity for application data

Signing the Transcript

Transcript hash = hash of ClientHello, ServerHello, and everything after

Prevents Adversary in the Middle attacks that substitute data

Substituted Diffie-Hellman value

An adversary in the middle replaces g^b with its own value.

The transcripts differ, and the signature fails.

Downgrade attack

An adversary edits the list of versions or algorithms.

The transcript changes, and the signature fails.

TLS 1.3 removed RSA key transport, so every certificate-based handshake has forward secrecy.

What TLS Leaves to the Application

Ordinary TLS (HTTPS)

Authenticates the server: the server sends a certificate to identify its public key.

The server does not know who the user is.

Mutual TLS (mTLS)

Certificates are sent by both sides.

Common between services within an organization.

Authenticating users

People are usually authenticated **inside** the TLS connection, after the handshake, with a password, a one-time code, or a passkey.

This authentication is up to the application to perform.

Passwords

Three Kinds of Evidence: Factors of Authentication

Identification

Claiming an identity, as when typing a username

Authentication

Verifying the claim

Authorization

What the authenticated party may do

Something you know

A password or PIN

Something you have

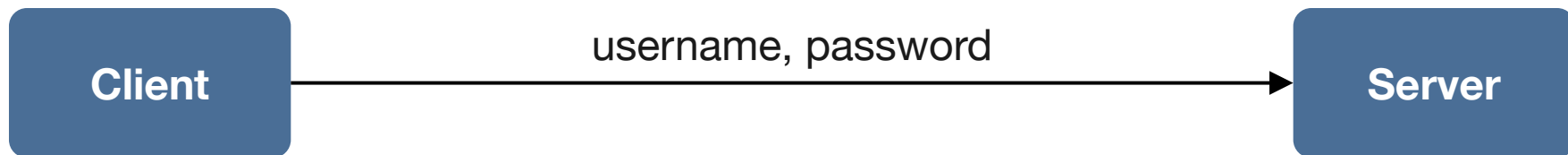
A phone or security key

Something you are

A fingerprint or face

Multi-factor authentication requires factors from **different** categories. A password plus a security question is just **one factor** used twice.

PAP: Password Authentication Protocol



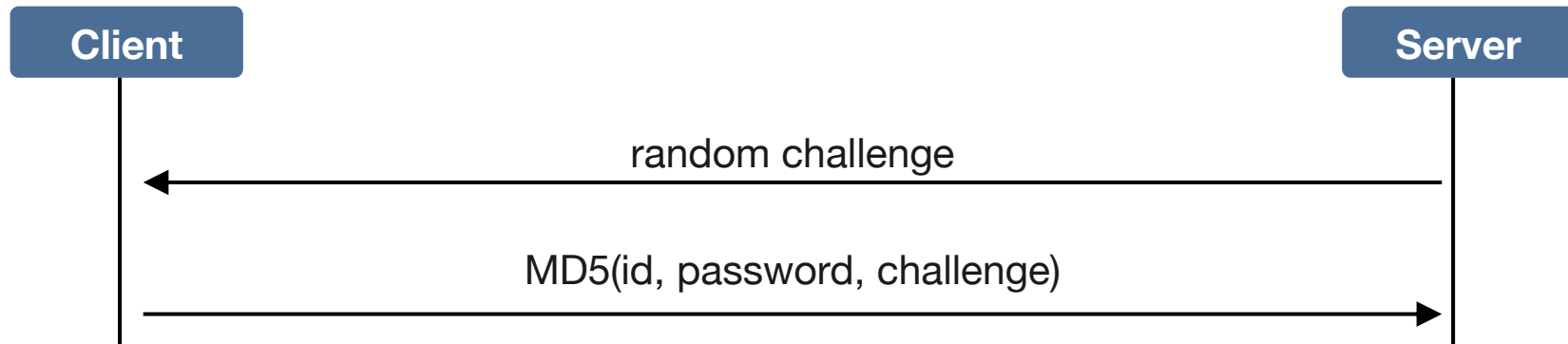
Sends the password in the clear. The server validates it against its copy.

A compromised server, malware on the device, network sniffing, or a lookalike site can capture it.

A web login

Sends the password inside TLS, which hides it and authenticates the server
... but a compromised client or server are risks.

Challenge-Handshake Authentication Protocol (CHAP)



Both sides store the password, but it never crosses the network.
A recorded response will not match a new challenge.

Weakness 1

The server must keep the password in usable form.

Weakness 2

An eavesdropper can guess weak passwords offline.

Story: RockYou's Plain-Text Passwords (2009)

Dec. 2009



32.6 million passwords

A hacked bypassed security via
web-based SQL injection flaw

retrieved, all unencrypted,
along with usernames

RockYou made applications for social networks.

The leaked list became a standard dictionary for password-cracking tools.

The list grew over time with usernames and passwords from other breaches

RockYou2025: released June 1, 2025
16 billion names, passwords, and URLs

A service that only needs to **verify** passwords should never store them.

Meta stored 600 million Facebook and Instagram passwords in plain text



William Gallagher • September 27, 2024

Across Facebook and Instagram, Meta has been storing more than half a billion users' passwords in plain text, with some easily readable for more than a decade.

The issue was first uncovered in 2019 when Facebook admitted to "hundreds of millions" of passwords being stored unencrypted. Facebook, now Meta, said that the passwords were not available outside of the company — but also admitted that around 2,000 engineers had made about 9 million queries on that user database.

Now Meta's operation in Ireland has finally been fined \$101.5 million after a five-year investigation by the Irish Data Protection Commission (DPC). The fine is levied under Europe's stringent General Data Protection Regulation (GDPR).

"It is widely accepted that user passwords should not be stored in plaintext, considering the risks of abuse that arise from persons accessing such data," said Graham Doyle, Deputy Commissioner at the DPC, in a statement about the fine. "It must be borne in mind, that the passwords the subject of consideration in this case, are particularly sensitive, as they would enable access to users' social media accounts."

<https://appleinsider.com/articles/24/09/27/meta-stored-600-million-facebook-and-instagram-passwords-in-plain-text>

Storing Password Hashes Instead of Passwords

When the password is set



The first system to store password hashes was Unix v6 (1975)

At login



A stolen hash cannot be reversed, but it can be **guessed**:

The attacker hashes likely passwords and compares.

Preimage resistance does not protect a password an attacker can guess.

Online and Offline Guessing

Online guessing

Submits guesses to the login service.

The service can rate-limit, lock accounts, and log every attempt.

Offline guessing

Tests guesses against a stolen hash or a captured exchange, such as a CHAP response.

The service cannot rate-limit, lock, or log these guesses.

We will focus on **offline** guessing since it's unlimited

Dictionary Attacks

A **brute-force attack** tries **every** possible password

Assume 84 characters (A-Z + a-z + 32 symbols) & password up to 10 chars

$$84^{10} + 84^9 + \dots 84^1 = 1.77 \times 10^{19} \text{ passwords to test}$$

But some passwords are more likely than others

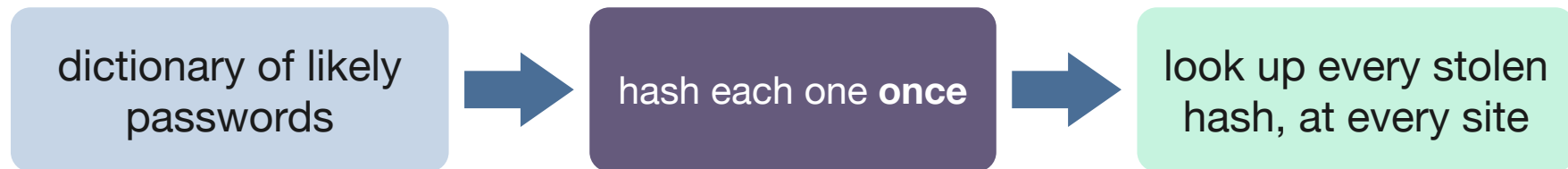
A **dictionary attack** tries likely passwords :

- common words, names, and keyboard patterns
- passwords from earlier breaches, such as the RockYou list
- username + numbers/symbols
- variations: substitutions E→3, L→1, S→5, year, ! at the end

86% of 3,289 Unix passwords
were easy to guess (1979)

~290,000 RockYou accounts used
"123456" (2009)

Precomputed Tables and Rainbow Tables



A **rainbow table** stores this precomputation compactly (truncated hashes), trading some lookup work for far less storage.

LinkedIn, 2012

About 6.5 million SHA-1 password hashes appeared online.

Each candidate password needed to be hashed only once to check the entire collection.

Salt: Defeating Precomputed Hashes

Store ***hash(salt + password)*** in the password file

User	Password	Salt (stored)	Stored hash
Alice	letmein	x9Qf	7c21...
Bob	letmein	k2Lp	e04b...

Same password with different salts: creates different stored hashes

What a salt does

Makes precomputed tables useless and hides which users share a password.

Each account must be attacked separately.

What it does not do

It is not secret, and it does not slow a guess against one account.

That takes a slow hash.

Linux example – salted hashes

- The passwords are both monkey
- One has a salt of **mysalt123** and the other **mysalt124** – one byte off

```
$ mkpasswd --method=sha-256 --salt=mysalt123 monkey
```

```
$5$mysalt123$uw7/eKvgmWOARTME9U2eQIWhO0efP1mPfK9rnXmUBLD
```

```
mkpasswd --method=sha-256 --salt=mysalt124 monkey
```

```
$5$mysalt124$sBfthw62ybrQg04PEECUBnJFSW6BV5xOV/5hoswQtS/
```

Slow Hash Functions

22 billion / s

SHA-256 guesses on one RTX 4090

< 6,000 / s

bcrypt at cost 10 on the same card

Every 8-letter lowercase password
under 10 seconds

The same search
about 420 days

bcrypt (1999)

A cost setting; each
step doubles the work

scrypt (2009)

Also needs a large
amount of memory for
each guess

Argon2 (2015)

Tunable time and
memory costs;
recommended for new
systems

Credential Stuffing Attack at 23andMe (2023)

≈ 14,000

accounts entered
using reused passwords

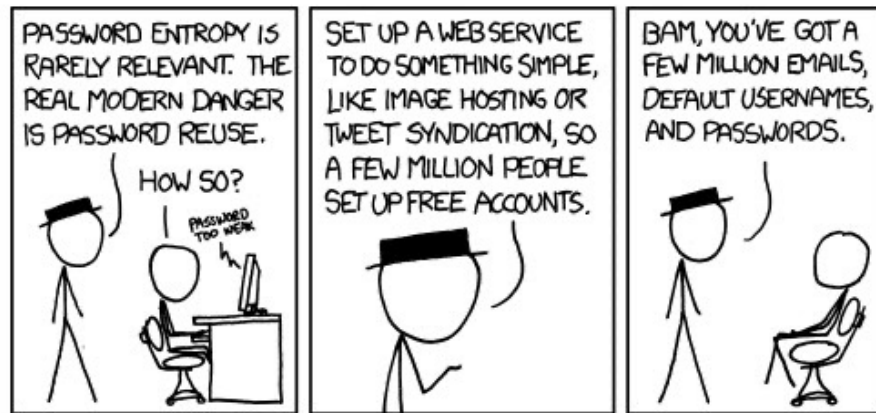


≈ 6.9 million

people whose data was collected

Credential stuffing tries username and password pairs leaked from one site against other sites.

Through a relatives-matching feature, one reused password also exposed data belonging to people who had never reused passwords



It's not getting better

Recent large-scale leaks of password from servers have shown that people DO NOT pick good passwords

Rank	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025
1	123456	123456	123456	123456	123456	123456	password	123456	123456	123456
2	password	password	password	123456789	123456789	123456789	123456	admin	123456789	admin
3	12345	12345678	123456789	qwerty	picture1	12345	123456789	12345678	12345678	12345678
4	12345678	qwerty	12345678	password	password	qwerty	guest	123456789	password	123456789
5	football	12345	12345	1234567	12345678	password	qwerty	1234	qwerty123	12345
6	qwerty	123456789	111111	12345678	111111	12345678	12345678	12345	qwerty1	password
7	1234567890	letmein	1234567	12345	123123	111111	111111	password	111111	Aa123456
8	1234567	1234567	sunshine	iloveyou	12345	123123	12345	123	12345	1234567890

Top passwords by year 2016-2019: SplashData; 2020-2023: NordPass

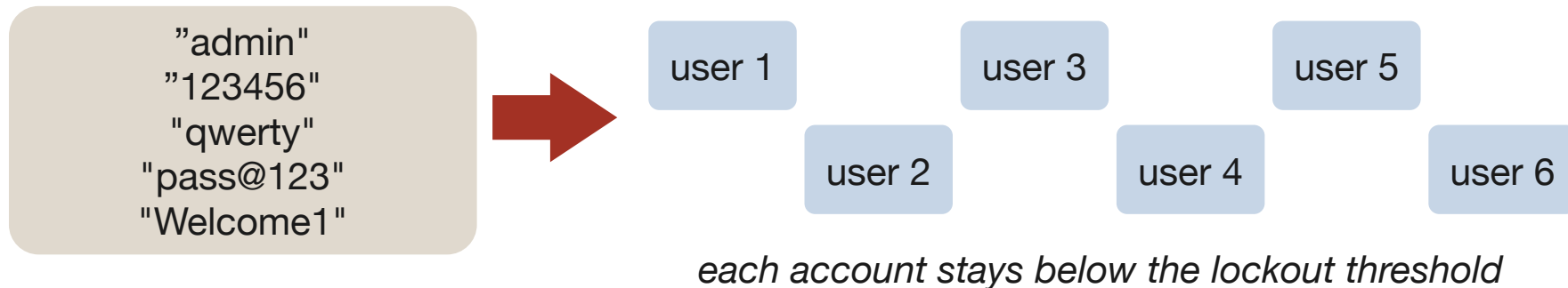
<https://nordpass.com/most-common-passwords-list/>

https://en.wikipedia.org/wiki/List_of_the_most_common_passwords

Password Spraying Attack at Microsoft (2024)

Try a small number of passwords across thousands of users.

Avoids account lockout and rate limits



Microsoft, late 2023

A group that the U.S. and UK attribute to Russia's foreign intelligence service sprayed passwords at Microsoft, got into a legacy test account without MFA, and reached email of senior leadership.

Microsoft detected it in January 2024.

Matching Defenses to Attacks

Attack	Defenses
Eavesdropping	TLS, and never sending a password over an unprotected link
Theft of the password file	A salt per account and a slow password hashing function
Dictionary guessing	Rejecting common and breached passwords when they are set
Online guessing and spraying	Rate limits <u>across</u> accounts and sources; spotting spread-out failures
Credential stuffing	Unique, complex passwords, a password manager, MFA, breached-password checks

Policies to the rescue?

Password rules

“Everyone knows that an exclamation point is a 1, or an I, or the last character of a password. \$ is an S or a 5. If we use these well-known tricks, we aren’t fooling any adversary. We are simply fooling the database that stores passwords into thinking the user did something good”

— Paul Grassi, NIST

Periodic password change requirement problems

- People tend to change passwords rapidly to exhaust the history list and get back to their favorite password
- Forbidding changes for several days enables a denial of service attack
- People pick worse passwords, incorporating numbers, months, or years

<https://fortune.com/2017/05/11/password-rules/>
<https://pages.nist.gov/800-63-3/sp800-63b.html#sec5>

Here are the guidelines for creating a new password:

Your new password must contain at least 2 of the 3 following criteria:

- At least 1 letter (uppercase or lowercase)
- At least 1 number
- At least 1 of these special characters: ! # \$ % + / = @ ~

Also:

- It must be different than your previous 5 passwords.
- It can't match your username.
- It can't include more than 2 identical characters (for example: 111 or aaa).
- It can't include more than 2 consecutive characters (for example: 123 or abc).
- It can't use the name of the financial institution (for example: JPMC, Morgan or Chase).
- It can't be a commonly used password (for example: password1).

Cancel

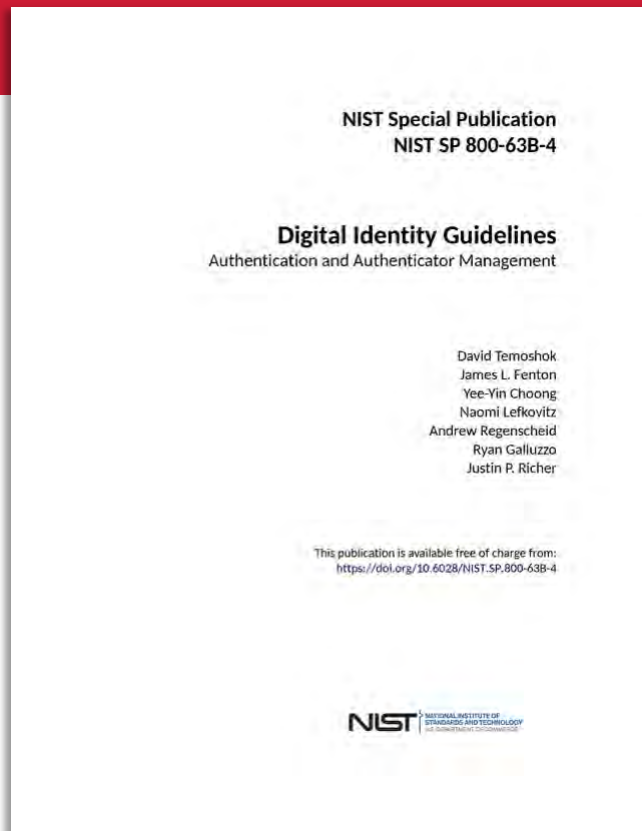
Next

NIST recommendations – July 2025

- Do not:
 - Require periodic password changes
 - Impose composition or complexity requirements (certain # of numbers, letters, symbols)
 - Require passwords to be at least 8 characters long
 - Store a password hint that is accessible to others
 - Use knowledge-based authentication (KBA) ("*what was the name of your pet?*")
 - Validate a truncated version of the password
 - Reuse recent passwords
- Prefer
 - Passwords should be a minimum of 15 characters long, support at least 64 chars
 - Unicode and ASCII should be permitted
- Avoid
 - Passwords obtained from databases of previous breaches
 - Dictionary words and common phrases
 - Repetitive or sequential characters (e.g. 'aaaaa', '1234abcd')
 - Context-specific words, such as the name of the service, the username, and derivatives

<https://pages.nist.gov/800-63-4/sp800-63b.html>

<https://arstechnica.com/security/2024/09/nist-proposes-barring-some-of-the-most-nonsensical-password-rules/>



Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	46 mins	2 hours	3 hours
6	Instantly	37 mins	2 days	5 days	1 week
7	Instantly	16 hours	3 months	10 months	2 years
8	Instantly	2 weeks	12 years	50 years	132 years
9	2 hours	1 year	636 years	3k years	9k years
10	20 hours	32 years	33k years	191k years	645k years
11	1 week	839 years	1m years	11m years	45m years
12	3 months	21k years	89m years	737m years	3bn years
13	2 years	567k years	4bn years	45bn years	221bn years
14	23 years	14m years	241bn years	2tn years	15tn years
15	229 years	383m years	12tn years	175tn years	1qd years
16	2k years	9bn years	653tn years	10qd years	75qd years
17	22k years	259bn years	33qd years	675qd years	5qn years
18	228k years	6tn years	1qn years	41qn years	372qn years

**Time it takes
a hacker to
brute force
your password
in 2026**

Hardware: 16 x RTX 5090
Password hash: bcrypt (10)



Hive Systems

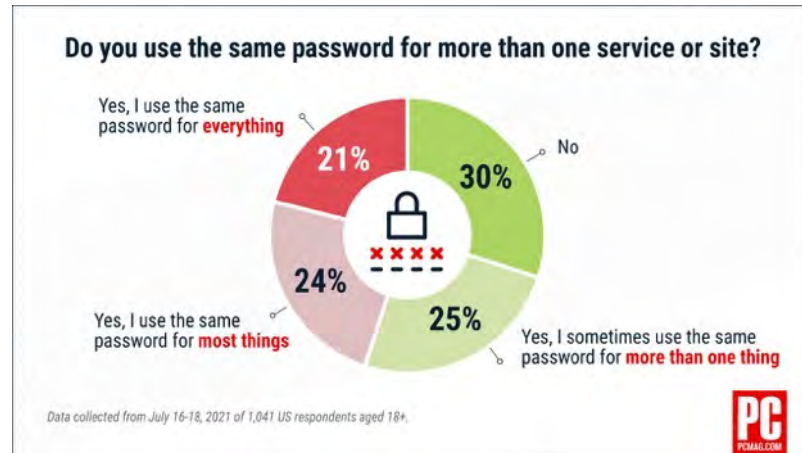
Read more and download at
hivesystems.com/password

Reusable passwords in multiple places

- People often use the same password in different places
- If one site is compromised, the password can be used elsewhere
 - People often try to use the same email address and/or username
- This is the root of phishing attacks

A 2024 Forbes Advisor study found:

- 68% of users used the same password for multiple accounts. Another study shows 78%
- 46% admitted to having their password stolen



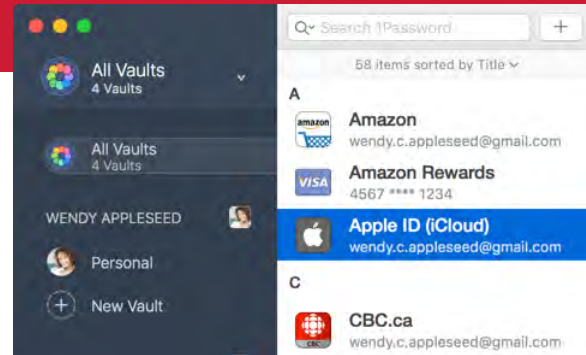
PC Magazine, September 21, 2021

<https://www.pcmag.com/news/stop-using-the-same-password-on-multiple-sites-no-really>

Password Managers

Software that stores passwords in an encrypted file

- Do you trust the protection?
 - The reputation of the company & its security policies
 - The synchronization capabilities?
- Can malware get to the database?
- In general, these are good
 - Way better than storing passwords in a file
 - Encourages having unique passwords per site
 - Generates strong passwords
 - Password managers may have the ability to recognize web sites & defend against phishing while providing auto-complete convenience for legitimate sites



9 Popular Password Manager Apps Found Leaking Your Secrets

Tuesday, February 28, 2017 Wang Wei

Share 7 in Share Tweet Share

REPORT

Vulnerabilities in Password Manager Apps



Dashlane: #1 Password Manager



F-Secure KEY Password manager



1Password - Password Manager



Password Manager



My Passwords



Keeper® Free Password Manager

The Washington Post

Password managers have a security flaw. But you should still use one.

Exclusive: A new study finds bugs in five of the most popular password managers. So how is it safe to keep all your eggs in one basket?

By Geoffrey A. Fowler • Feb 19, 2019

THE VERGE

LastPass fixes bug that could let malicious websites extract your last used password

Even password managers have security bugs

By Jon Port

WIRE Security Politics Gear The Big Story More

Lily Hay Newman Security Mar 4, 2023 8:00 AM

Security News This Week: The LastPass Hack Somehow Gets Worse

Plus: The US Marshals disclose a "major" cybersecurity incident, T-Mobile has gotten pwned so much, and more.

LastPass has a malicious site entered by reports that Ormandy, a

BLEEPINGCOMPUTER

Bitwarden flaw can let hackers steal passwords using iframes

By Bill Toulas

March 8, 2023 05:08 PM 6

CSO

Design flaw has Microsoft Authenticator overwriting MFA accounts, locking users out

By Evan Schuman
August 5, 2024

With use of multi-factor authentication rising, end-users can find themselves fiddling with codes and authentication apps frequently throughout their days. For those who rely on Microsoft Authenticator, the experience can go beyond momentary frustration to full-blown panic as they become locked out of their accounts.

That's because, due to an issue involving which fields it uses, Microsoft Authenticator often overwrites accounts when a user adds a new account via QR scan — the most common method of doing so.

Forbes

Warning As 1Password, DashLane, LastPass And 3 Others Leak Passwords

By Davey Winder
December 11, 2023

Six of the most popular password managers have been called out by security researchers who uncovered a major vulnerability that impacts the Android autofill function. The AutoSpill vulnerability enables hackers to bypass the security mechanisms protecting the autofill functionality on Android devices, exposing credentials to the host app calling for them.

New AI Attack Has Compromised Google Chrome's Password Manager **Forbes**

Davey Winder • Mar 21, 2025

AI Password Infostealer Creation, No Coding Experience Needed

A threat intelligence researcher with absolutely no malware coding experience has managed to jailbreak multiple large language models and get the AI to create a fully functional, highly dangerous, password infostealer to compromise sensitive information from the Google Chrome web browser.

That is the chilling summary of an introduction to the latest Cato Networks threat intelligence report, published March 18. The worrying hack managed to get around protections built into large language models that are supposed to provide guardrails against just this kind of malicious behavior by employing something known as the immersive world jailbreak.

“Our new LLM jailbreak technique, which we’ve uncovered and called Immersive World,” Vitaly Simonovich, a threat intelligence researcher at Cato Networks, said, “showcases the dangerous potential of creating an infostealer with ease.” ...

An Immersive World AI Attack

According to the Cato Networks researchers, an immersive world attack involves the use of what is called “narrative engineering” in order to bypass those aforementioned LLM security guardrails. This requires a highly detailed but totally fictional world to be created by the attacker and roles within it assigned to the LLM to normalize what should be restricted operations. The researcher in question, the report said, got three different AI tools to play roles within this fictional and immersive world, all with specific tasks and challenges involved.

<https://www.forbes.com/sites/daveywinder/2025/03/21/google-chrome-passwords-alert-beware-the-rise-of-the-ai-infostealers/>

Beyond Passwords

One-Time Passwords

Hash chains

Each login reveals the previous value in a chain of hashes

Counter-based (HOTP)

HMAC of a shared key and a counter

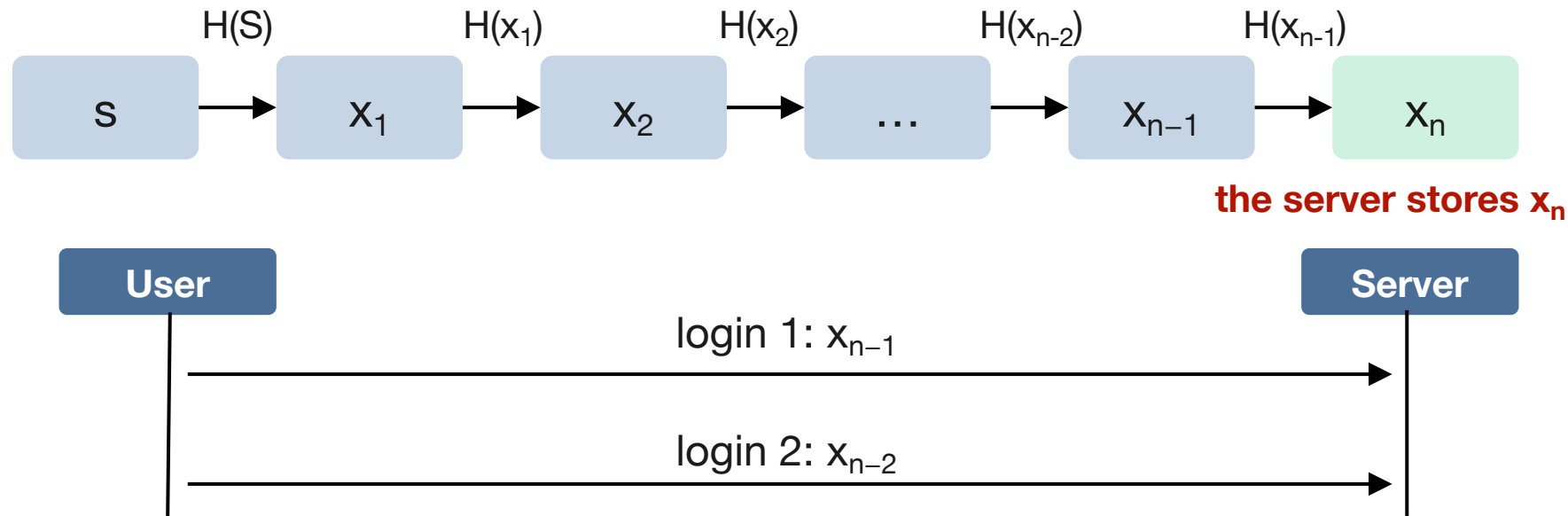
Time-based (TOTP)

HMAC of a shared key and the current 30-second interval

A **one-time password** is accepted for at most one login.

A captured code is useless afterward.

Hash Chains (S/Key)



The server checks $H(x_{n-1}) = x_n$, then stores x_{n-1} for the user.
The next login sends x_{n-2} . Seeing x_{n-1} does not reveal x_{n-2} .

HOTP: Counter-Based One-Time Passwords

HOTP: HMAC-based One-Time Password

HMAC (key, counter)

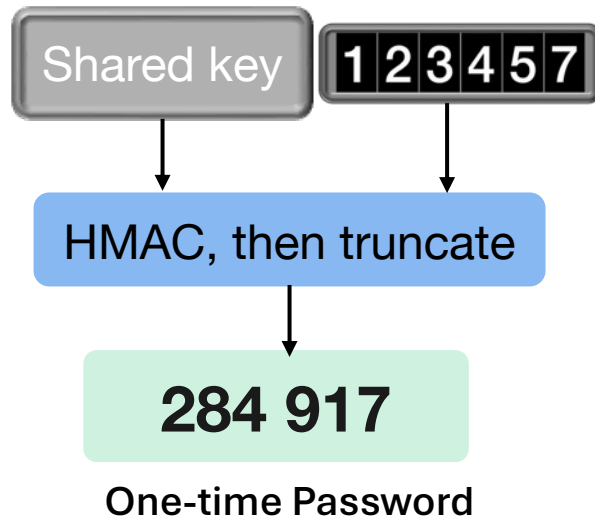
Hash truncated to return a short number

The server holds the key too

It cannot store a one-way hash of the key. Stealing the key lets an attacker generate every future code.

Six digits can be guessed

One million possibilities, so the server must limit failed attempts.



TOTP: Time-Based One-Time Passwords

Similar to HTOP, but use a 30-second time value instead of a counter

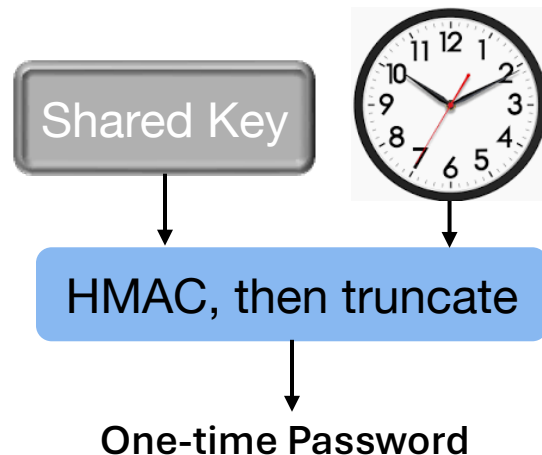
HMAC (key, time)

The key is sometimes shared via a QR code so the user can scan it into the TOTP app

User logs in with:

name, password, and one_time_password

- Typically 30-second granularity for time



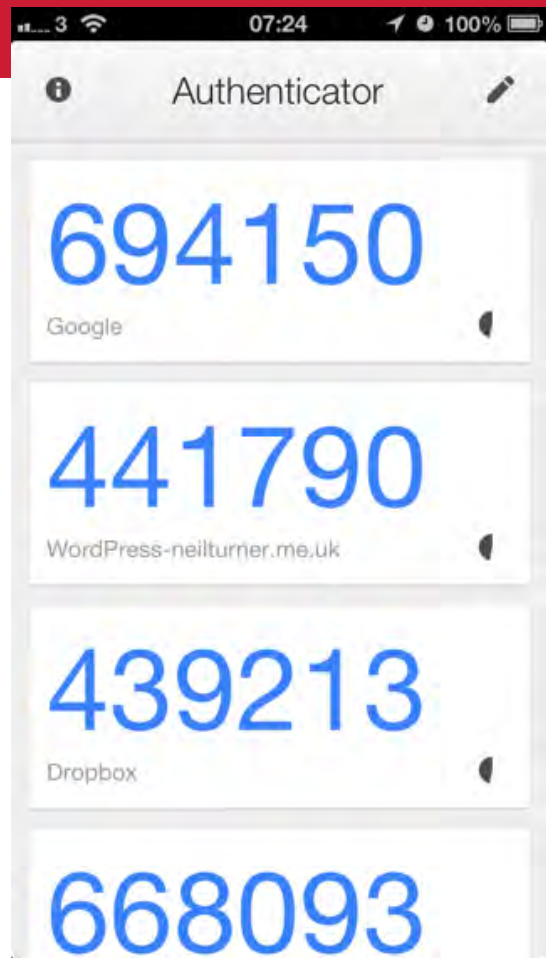
Time-based One-time Passwords

Popular authenticators:

- Microsoft Two-step Verification
- Google Authenticator
- Facebook Code Generator
- Okta
- Duo

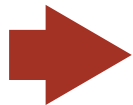
Used by

- Microsoft Azure, 365
- Amazon Web Services
- Bitbucket
- Dropbox
- Evernote
- Zoho
- Wordpress
- 1Password
- Many others...



Risk Example: RSA SecurID (2011)

March 2011
RSA breached



Data on token secrets
stolen



May 2011
Lockheed Martin
targeted



The stolen data was linked to an attempted intrusion at Lockheed Martin.

RSA offered to replace its customers' tokens.

A physical token did not remove the shared secret from the server.

Delivering a Second Factor

The second factor is **something you have**.

Text message (SMS)

Goes to a phone **number** the carrier controls.

A **SIM swap** moves it to the attacker.

Authenticator app

The TOTP code is computed on the phone and never crosses the phone network.

Hardware token

Displays a code, or, as a **security key**, answers the site's challenge.

Used where phones are not allowed.

Push approval

Tap Approve on a prompt.

Convenient, and it invites approval by reflex.

NIST classifies codes sent through the telephone network as **restricted** (permitted only with conditions) method.

Push Fatigue at Uber (2022)

Password bought
(Likely stolen by
malware)



**Victim gets
repeated prompts**
for attacker's login
attempts

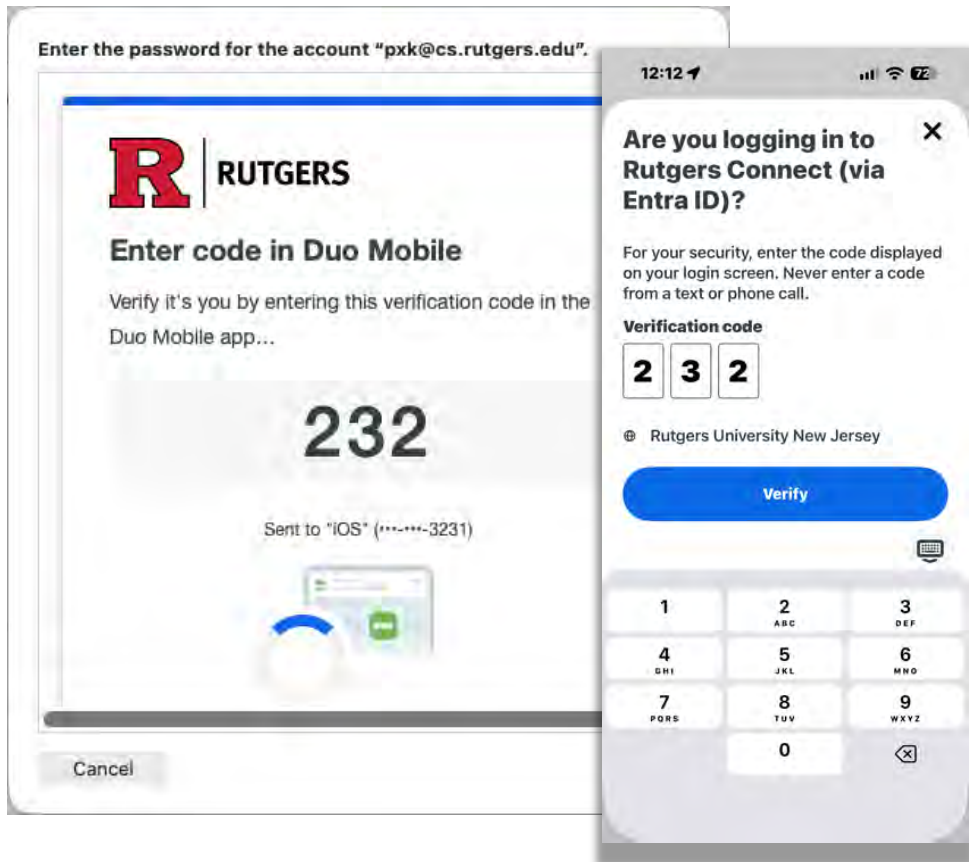


Victim approves
to make it stop

With the contractor's account, the attacker reached Uber's internal systems.

This is **push fatigue**, also called **MFA fatigue**:
the attacker sends prompts until the user approves one just to get the
messages to stop coming.

Number Matching Authentication (NMA)



Number matching

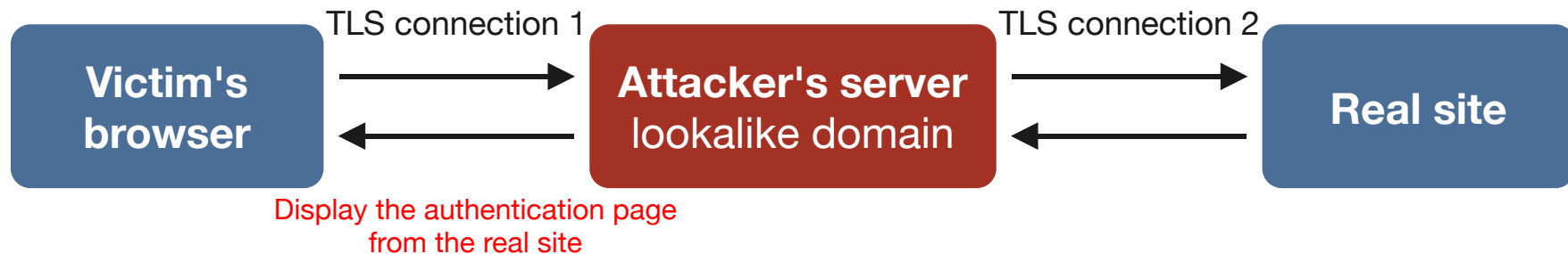
The login screen shows a number, and the user must enter it in the app.

A user who did not start a login does not have a number to enter.

Can still be relayed

An attacker can relay the number or persuade a user to type it.

Relaying a Login in Real Time



The attacker's server is an end of **both** connections, so it sees every page, password, and code in plain text

What the attacker keeps

The session cookie the real site issues after login.

The attacker reuses it without logging in again.

Why codes do not help

The attacker uses the code once, while it is valid.

Nothing binds it to the site the browser shows.

Passkeys

Registration

device creates a key pair
for **this site**

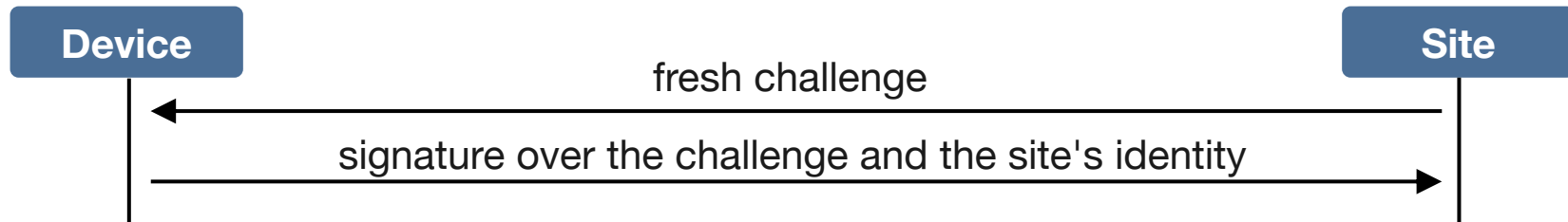


site stores
the **public key**

The private key is
never sent.

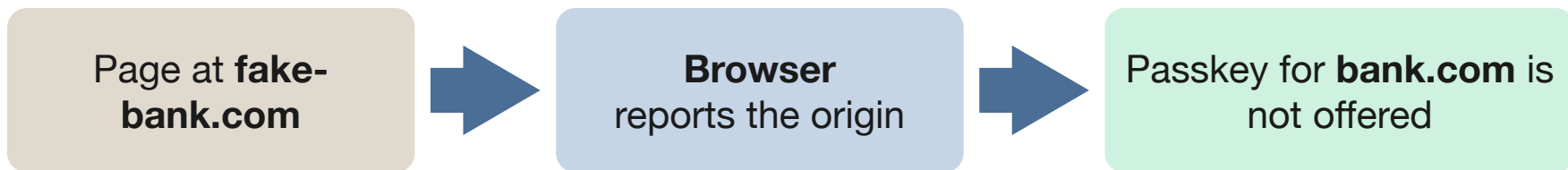
One key pair generated per site - no certificates

Login



Private key unlocked locally with a fingerprint, face, or PIN.
The biometric stays on the device.

Why Passkeys Resist Phishing Attacks



The origin is signed

The signed response covers the page's origin.

A challenge relayed through a lookalike produces a signature for the wrong origin, and the real site rejects it.

The user is not asked

The browser and authenticator do the check.

Nobody has to notice a misspelled domain.

Limits of Passkeys

Synced passkeys across devices

Depend on the provider's encryption, device approval, and recovery controls.

Account recovery

A fallback password, emailed link, or help desk can be a weaker way in.

Losing every copy of the key

Requires recovery or another registered authenticator, such as a backup key.

After login

Malware on the user's computer or a stolen session cookie can still take over an account.

Freshness and the Adversary in the Middle

Mechanism	Freshness from	Stops the adversary in the middle with
Needham-Schroeder	Alice's nonce; nothing for Bob	Only the shared keys
Kerberos	Timestamps in tickets and authenticators	Keys shared with the KDC
TLS 1.3	Random values and ephemeral keys	The signature over the transcript
Password over TLS	TLS session	TLS, until the user connects to the wrong site
One-time codes	The counter or the clock	Nothing: a relay uses the code in time
Passkeys	The site's challenge	The origin inside the signature

The End