PV204 Security technologies

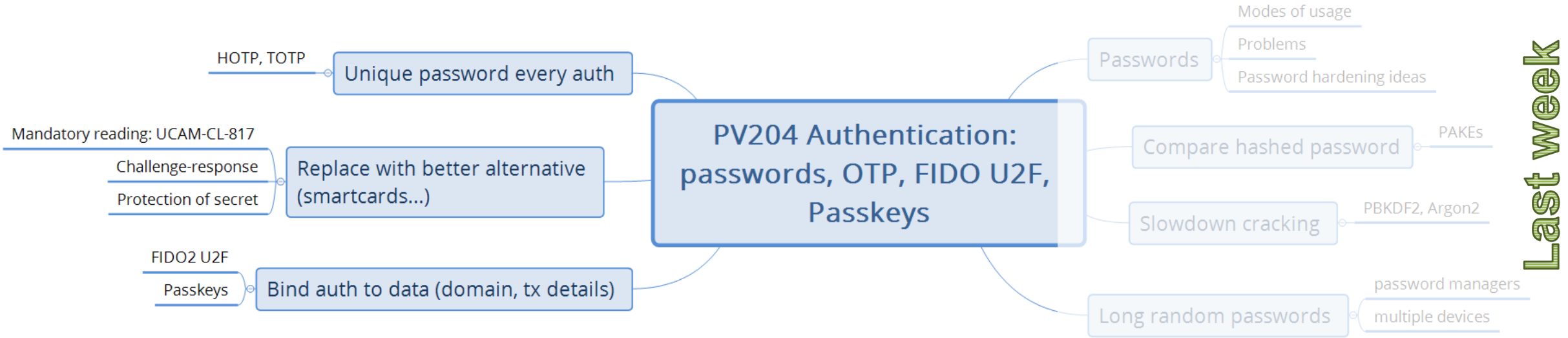
Authentication and secure channels II.

FIDO2, Passkeys, ECDH, Forward secrecy, KEM, PAKE, PQC

Petr Švenda *svenda@fi.muni.cz* *@rngsec*
Centre for Research on Cryptography and Security, Masaryk University

CROCS
Centre for Research on Cryptography and Security

Please provide any corrections and comments here (thank you!):
https://drive.google.com/file/d/1xtewmdqFVmQtxr_9zq75I_-hLNp4dq_J/view?usp=sharing


PV204 Authentication: passwords, OTP, FIDO U2F, Passkeys
Unique password every auth
HOTP, TOTP
Replace with better alternative (smartcards...)
Mandatory reading: UCAM-CL-817
Challenge-response
Protection of secret
Bind auth to data (domain, tx details)
FIDO2 U2F
Passkeys
Passwords
Modes of usage
Problems
Password hardening ideas
Compare hashed password
PAKEs
Slowdown cracking
PBKDF2, Argon2
Long random passwords
password managers
multiple devices
Last week

Recall: Password “hardening” ideas

Last week

1. Hash password by one-way function (hard(er) to invert)
2. Slowdown cracking attempts (less potential passwords tried)
3. Enable users to have long, random and unique passwords
4. Have unique password for every authentication attempt
5. Replace/complement passwords with something else (e.g., smartcard)
6. Bind response to server domain name (to prevent phishing)

IDEA: HAVE UNIQUE PASSWORD FOR EVERY AUTHENTICATION ATTEMPT

ONE-TIME PASSWORDS: HOTP & TOTP

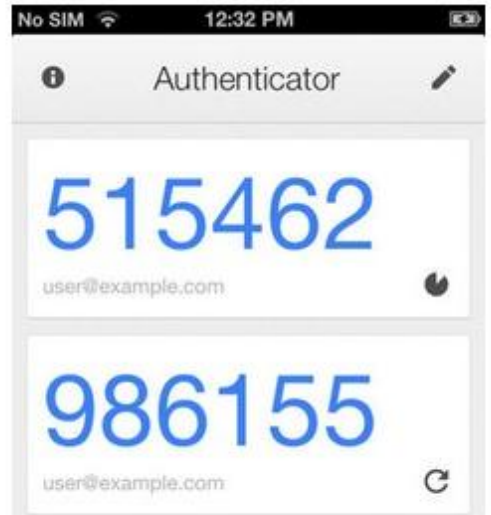
Recall: Problems associated with passwords

- How to create secure password?
- How to use password securely?
- How to store password securely?
- Same value is used for the long time (exposure)
- Value of password is independent from target operation (e.g., authorization of request)
- …

One-time passwords tries to address these issues

HMAC-based One-time Password Algorithm (RFC 4226)

- HMAC-based One-time Password Algorithm (HOTP)
  - Secret key K
  - Counter (challenge) C
  - $HMAC(K,C) = SHA1(K \oplus 0x5c5c\ldots \| SHA1(K \oplus 0x3636\ldots \| C))$
  - $HOTP(K,C) = Truncate(HMAC(K,C)) \& 0x7FFFFFFF$
  - 0x7FFFFFFF mask to drop most significant bit (portability)
  - $HOTP\text{-}Value = HOTP(K,C) \bmod 10^d$ (d … # of digits)
- Many practical implementations
  - E.g., Google Authenticator
- https://en.wikipedia.org/wiki/HOTP



HOTP – items, operations

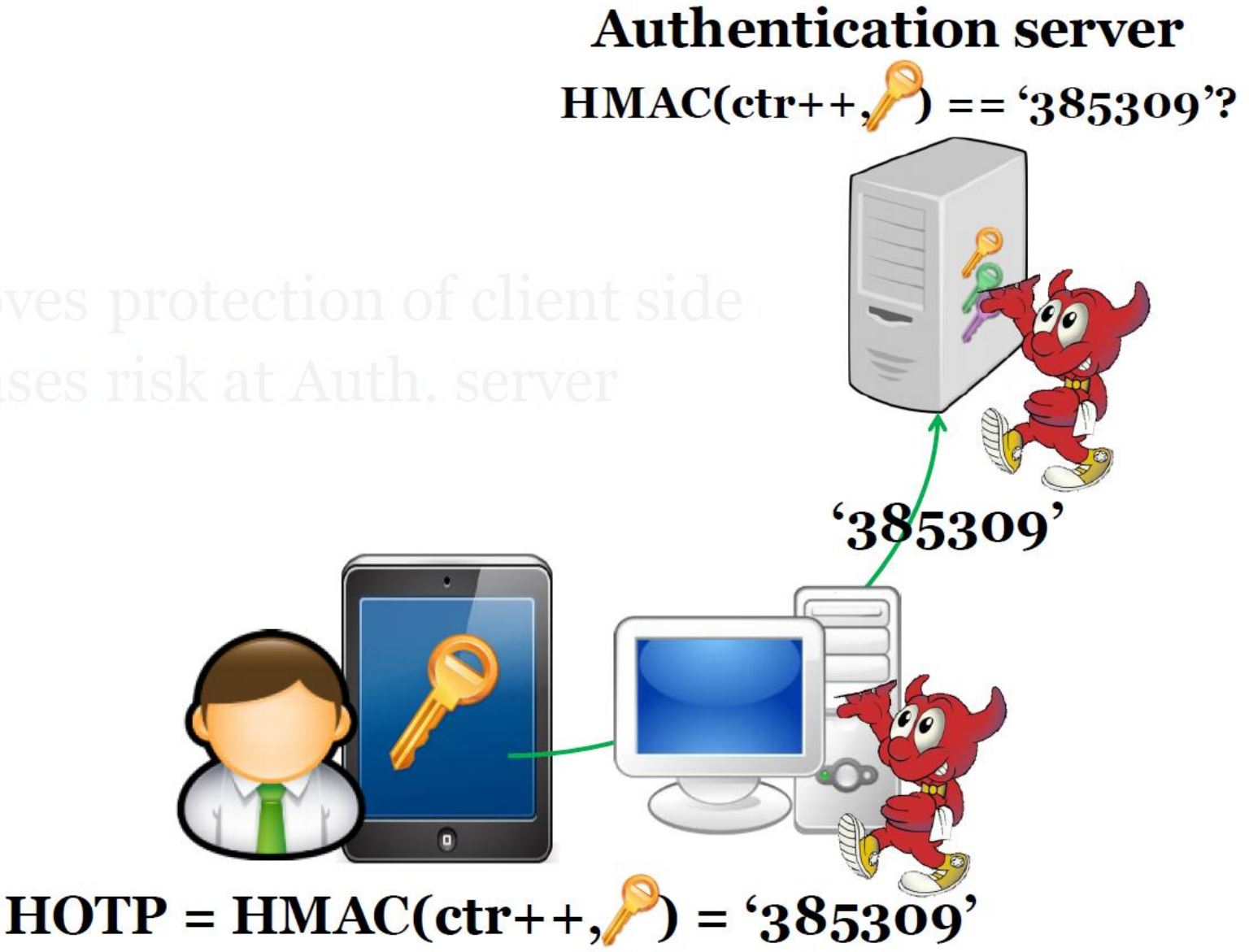
- Logical operations
  1. Generate initial state for new user and distribute master key
  2. Generate HOTP code and update state (user)
  3. Verify HOTP code and update state (auth. server)
- Security considerations of HOTP
  - Client compromise (=> OTP master key compromised)
  - Server compromise (=> OTP master key compromised)
  - Repeat of counter/challenge (already used OTP code checked)
  - Counter mismatch tolerance window (counter mismatch == OTP code mismatch)
  - Phishing: user enters HOTP code at phishing website, attacker resent to real one

Time-based One-time Password Algorithm

- Very similar to HOTP
  - Time used instead of counter
- Requires synchronized clocks
  - In practice realized as time window
- Tolerance to gradual desynchronization possible
  - Server keeps device's desynchronization offset
  - Updates with every successful login

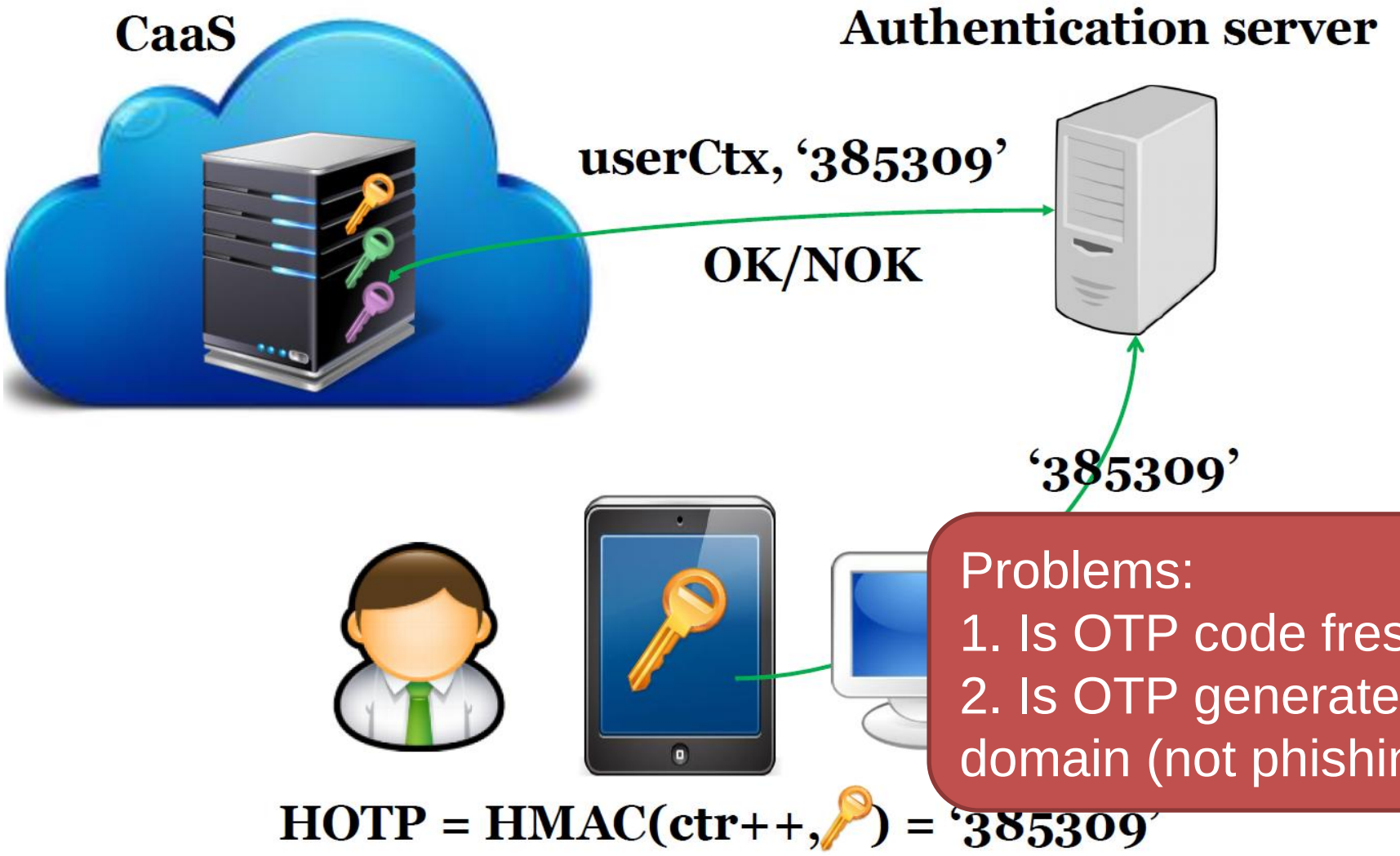
OCRA: OATH Challenge-Response Algorithm

- Initiative for Open Authentication (OATH)
- OCRA is authentication algorithm based on HOTP
- OCRA code = CryptoFunction(K, DataInput)
  - *K*: a shared secret key known to both parties
  - *DataInput*: concatenation of the various input data values
    - Counter, challenges, H(PIN/Passwd), session info, H(time)
  - Default *CryptoFunction* is HOTP-SHA1-6
  - https://tools.ietf.org/html/rfc6287
- Don’t confuse with OAuth (delegation of authentication)
  - The OAuth 2.0 Authorization Framework (RFC6749)
  - TLS-based security protocol for accessing HTTP service


Authentication server
HMAC(ctr++, 🔑) == '385309'?
'385309'
HOTP = HMAC(ctr++, 🔑) = '385309'

Increased risk at *OTP verification server

- More secure against client compromise
  - Using OTP instead of passwords, KDF(time|key),
- But what if server is compromised?
  - database hacks, temporal attacker presence
  - E.g., Heartbleed => dump of OTP keys from server memory
- Possible solution: Trusted hardware on the server
  - OTP code verified inside trusted environment
  - OTP key never leaves the hardware

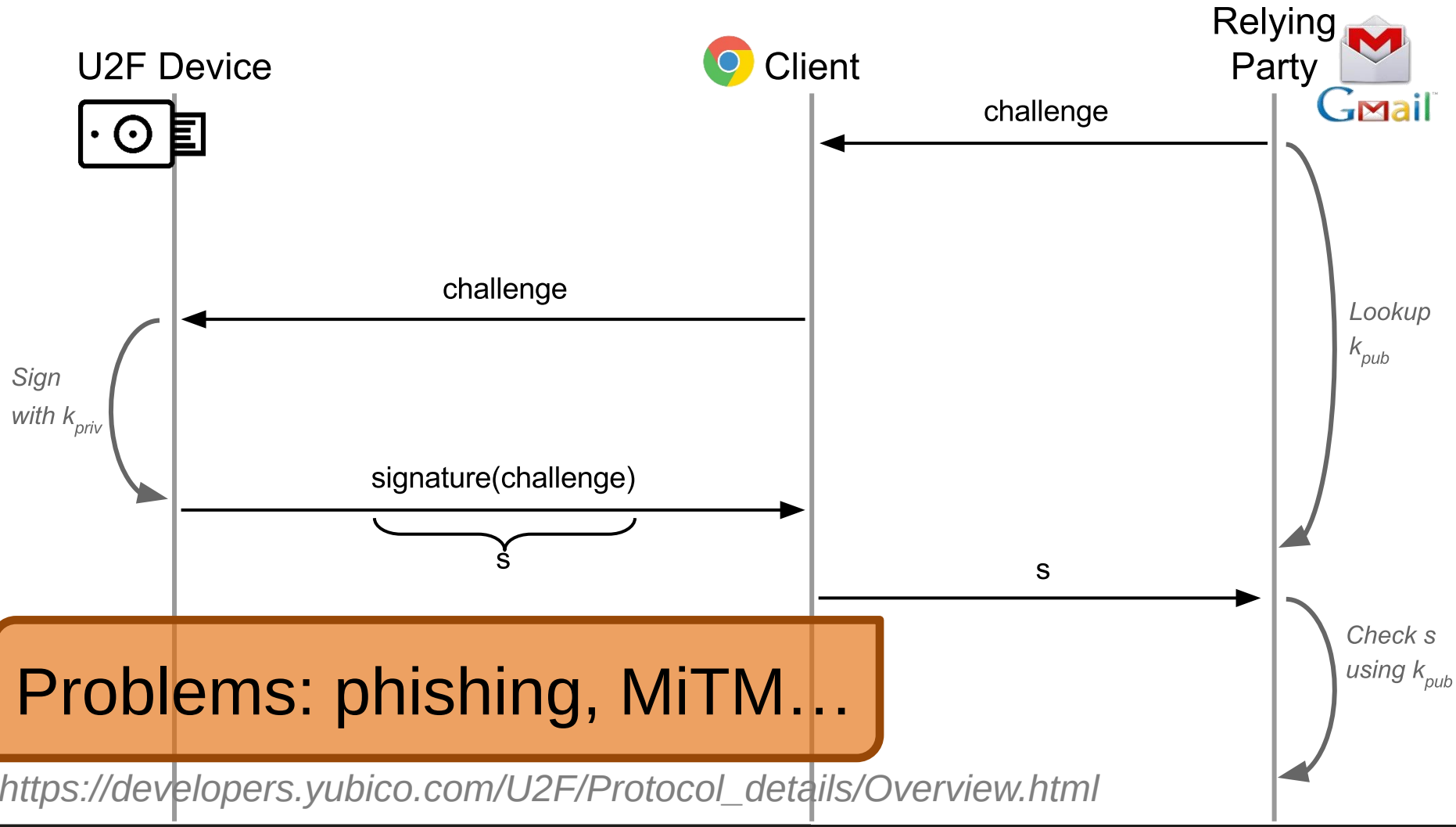


Problems:
1. Is OTP code fresh?
2. Is OTP generated for correct domain (not phishing attempt)?

IDEA: REPLACE PASSWORD BY SMARTCARD WITH ASYMMETRIC KEYPAIR, CHALLENGE-RESPONSE PROTOCOL AND PREVENT PHISHING

FIDO U2F PROTOCOL (U2F → FIDO2 → WEBAUTHN)

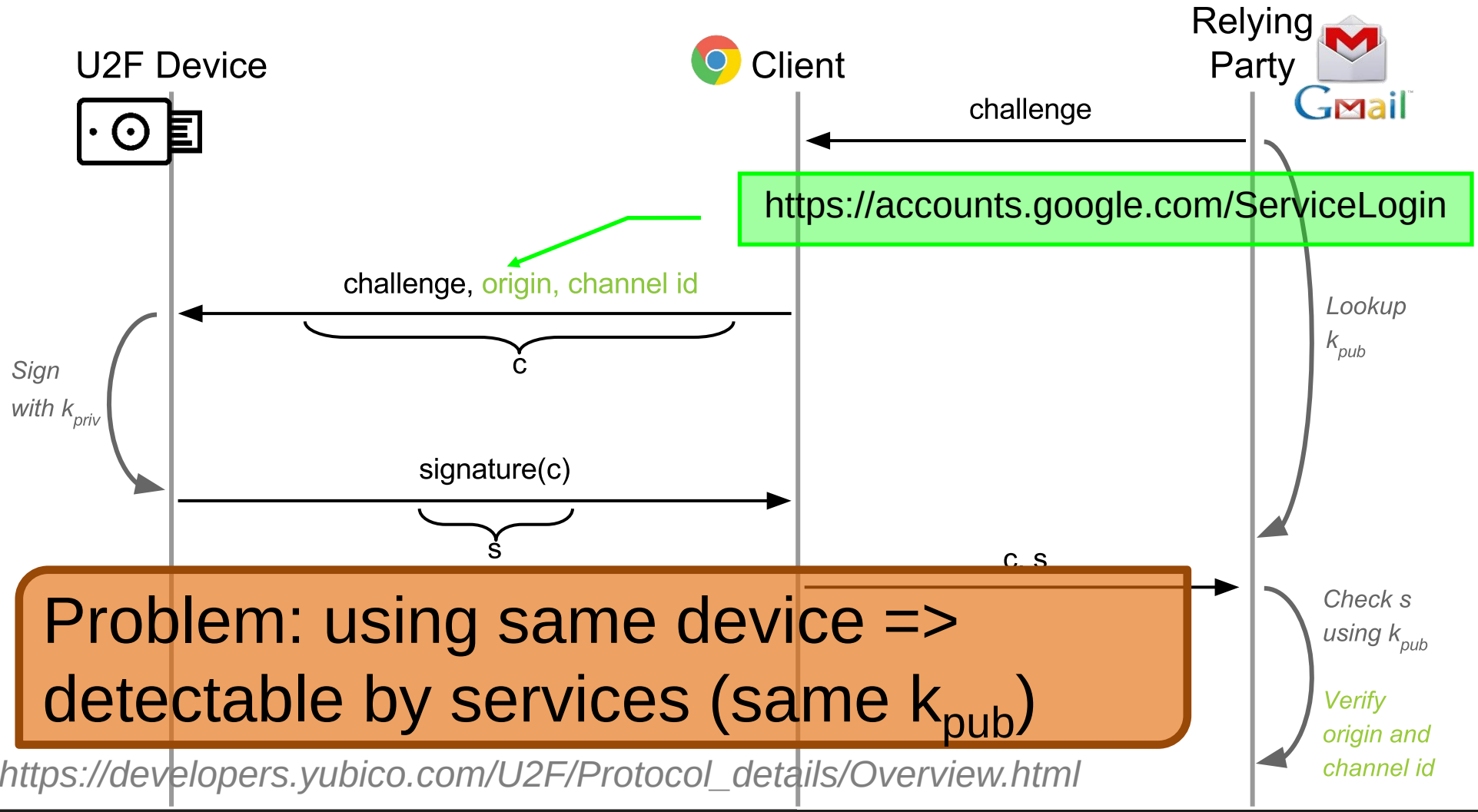
Revision 1: ECC-based challenge-response



Problems: phishing, MiTM…

https://developers.yubico.com/U2F/Protocol_details/Overview.html

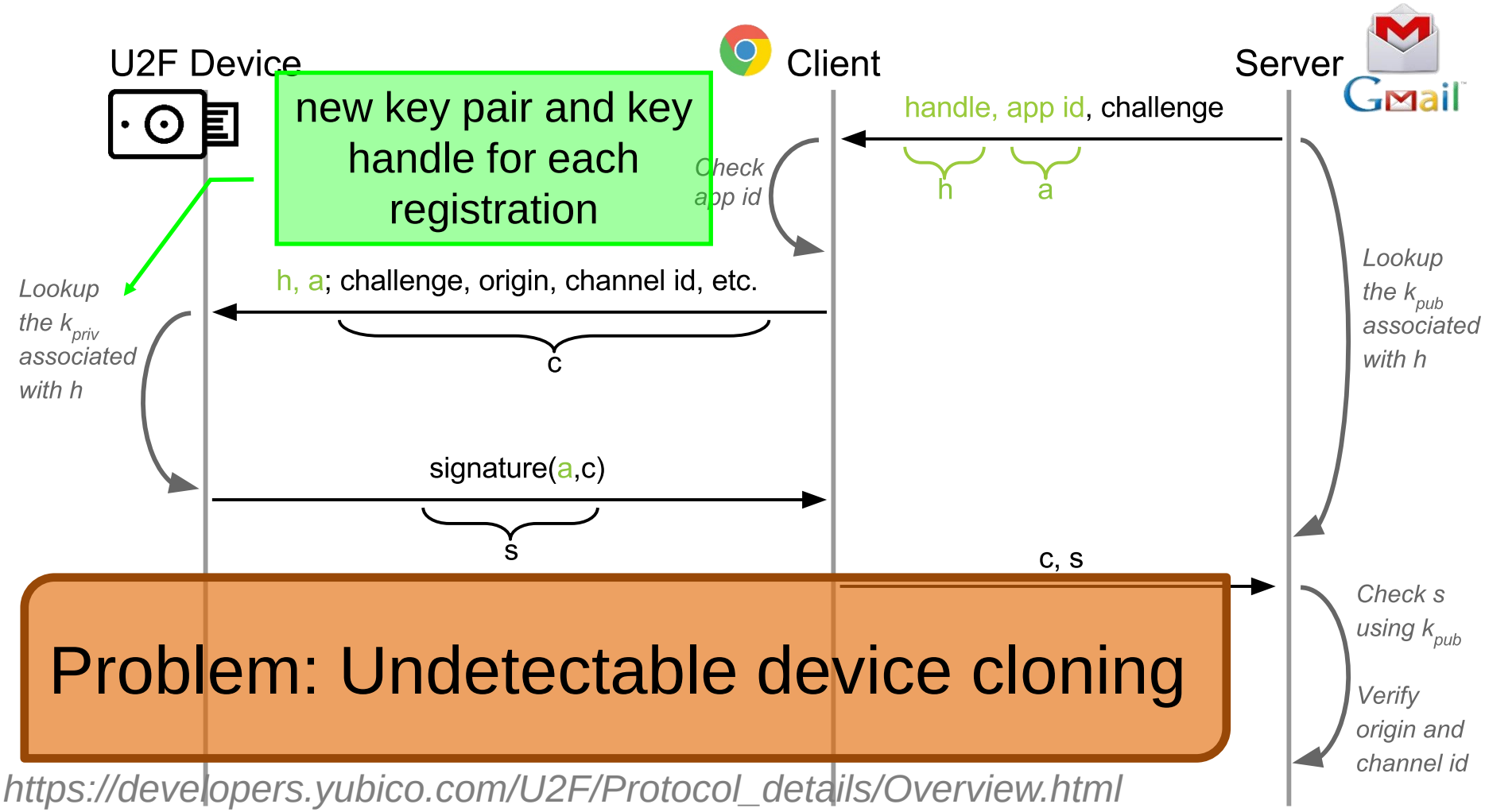
Revision 2: URI + TLS channel id added



Problem: using same device => detectable by services (same k_{pub})

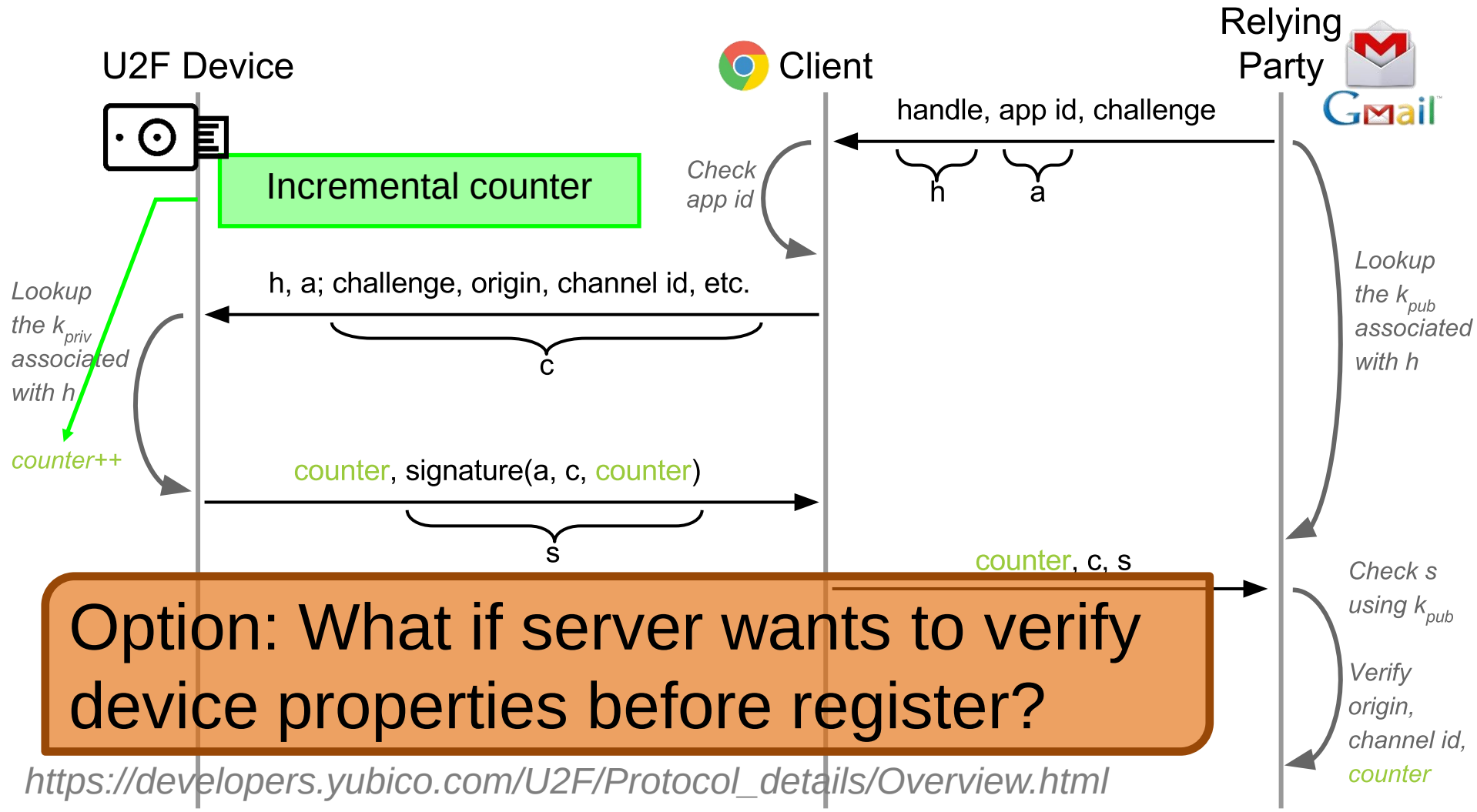
https://developers.yubico.com/U2F/Protocol_details/Overview.html

Revision 3: Application-specific key added



https://developers.yubico.com/U2F/Protocol_details/Overview.html

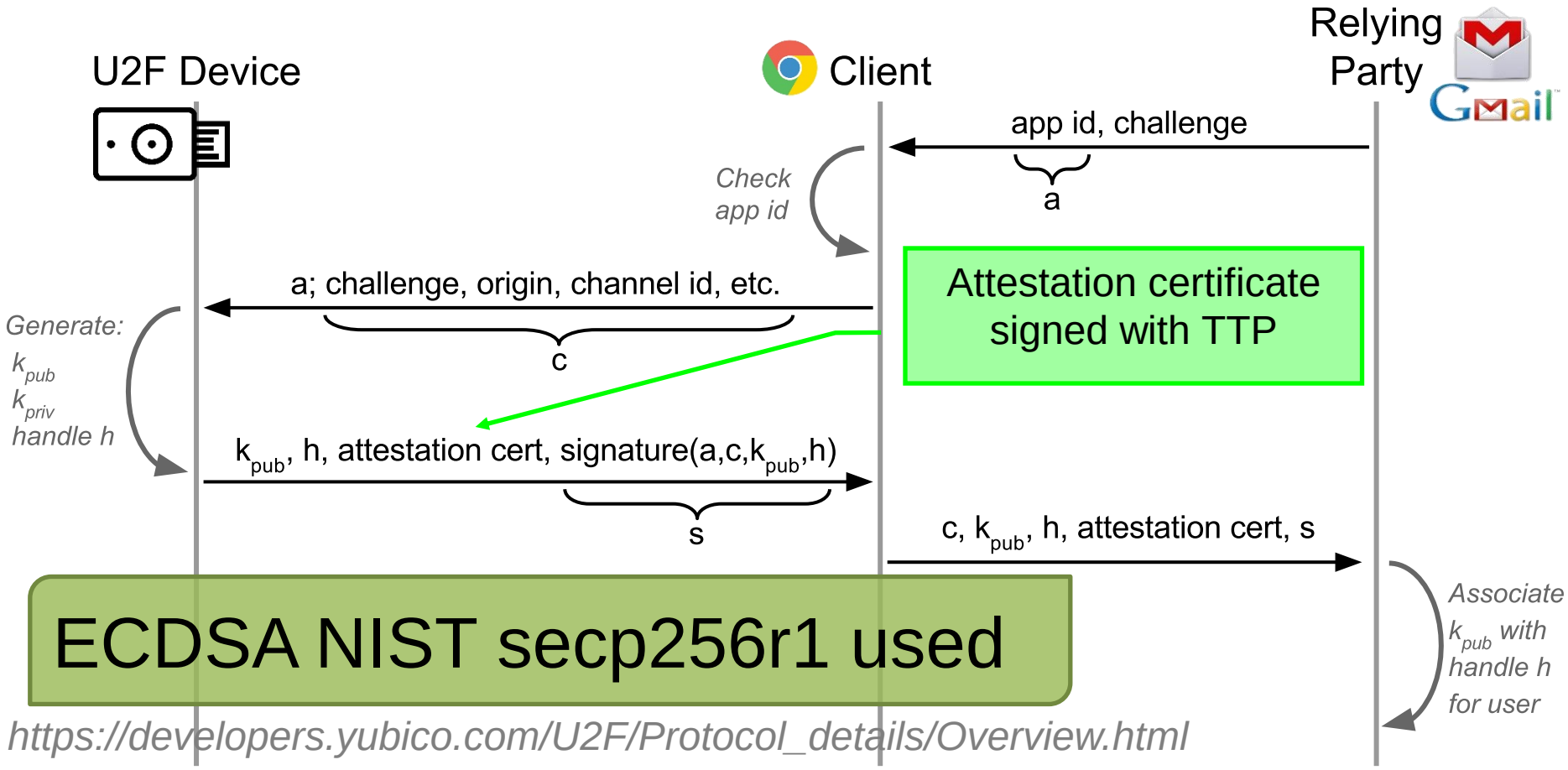
Revision 4: Authentication counter added



Option: What if server wants to verify device properties before register?

https://developers.yubico.com/U2F/Protocol_details/Overview.html

Revision 5: Device attestation added



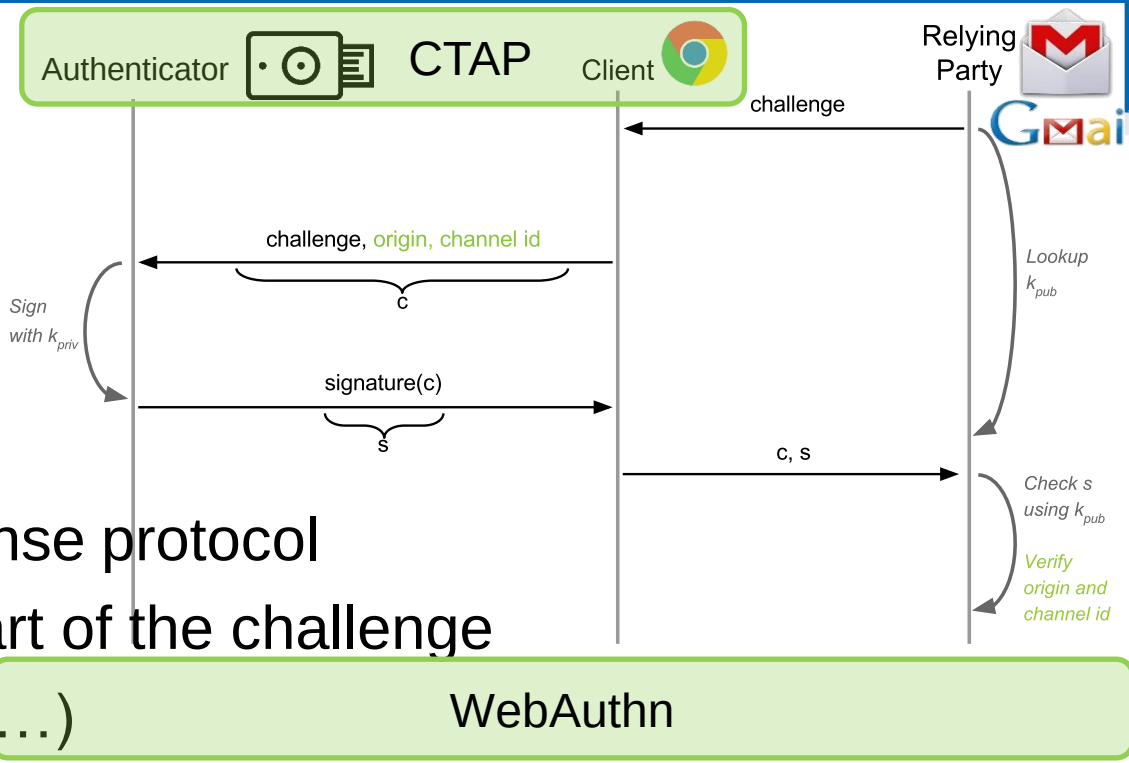
ECDSA NIST secp256r1 used

https://developers.yubico.com/U2F/Protocol_details/Overview.html

U2F/FIDO2/WebAuthn – current state

- FIDO alliance of major companies
- U2F → FIDO2 → WebAuthn (more than “just” U2F)
- Original U2F protocol extended and moved under W3 as WebAuthn
  - https://www.w3.org/TR/webauthn/
- Large selection of tokens now available (including open-hardware)
- Android, iPhone added systematic support for FIDO U2F (02/2019)
  - Android phone acts as U2F token
  - https://www.wired.com/story/android-passwordless-login-fido2
  - Google Smart Lock app on iOS uses secure enclave and acts as FIDO token
  - Since iOS 13.3. USB, NFC, and Lightning FIDO2-compliant security keys in Safari browser (12/2019))

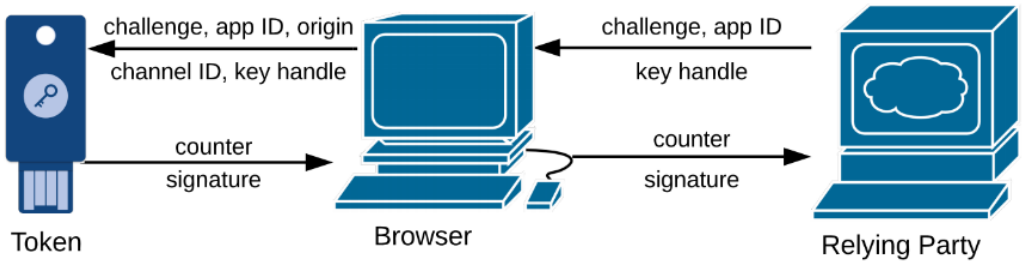
CTAP/WebAuthn stack



- WebAuthn Protocol
  - Asymmetric crypto-based challenge-response protocol
  - Browser inserts actual URL (origin) as a part of the challenge
  - Private key stored and used (token, phone…)
  - An API for accessing Public Key Credentials Level 2 (level=version)
    - Official documentation: https://www.w3.org/TR/webauthn/
- Client to Authenticator Protocol (CTAP)
  - Protocol between browser and authenticator
  - Authenticator = initially hardware token, but now range of devices (phones, calculators…)

https://developers.yubico.com/U2F/Protocol_details/Overview.html

True2F FIDO U2F token



- Yubikey 4 has single master key
  - To efficiently derive keypairs for separate Relying parties (Google, GitHub…)
  - Inserted during manufacturing phase (what if compromised?)
- Additional SMPC protocols (protection against backdoored token)
  - Secure Multi-Party Computation (SMPC) will be covered later
  - Verifiable insertion of browser randomness into final keypairs
  - Prevention of private key leakage via ECDSA padding

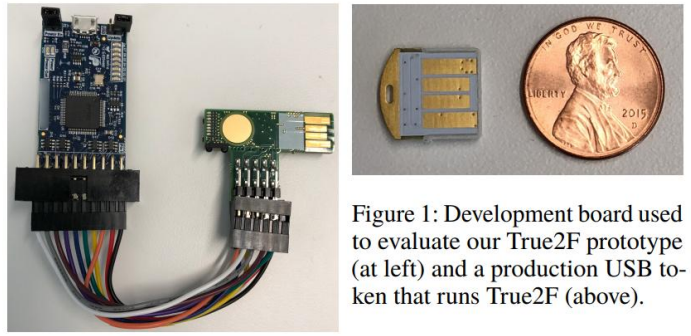


Figure 1: Development board used to evaluate our True2F prototype (at left) and a production USB token that runs True2F (above).

- Backward-compatible (Relying party, HW)
- Efficient: 57ms vs. 23ms to authenticate

WebAuthn - evolution of U2F protocol

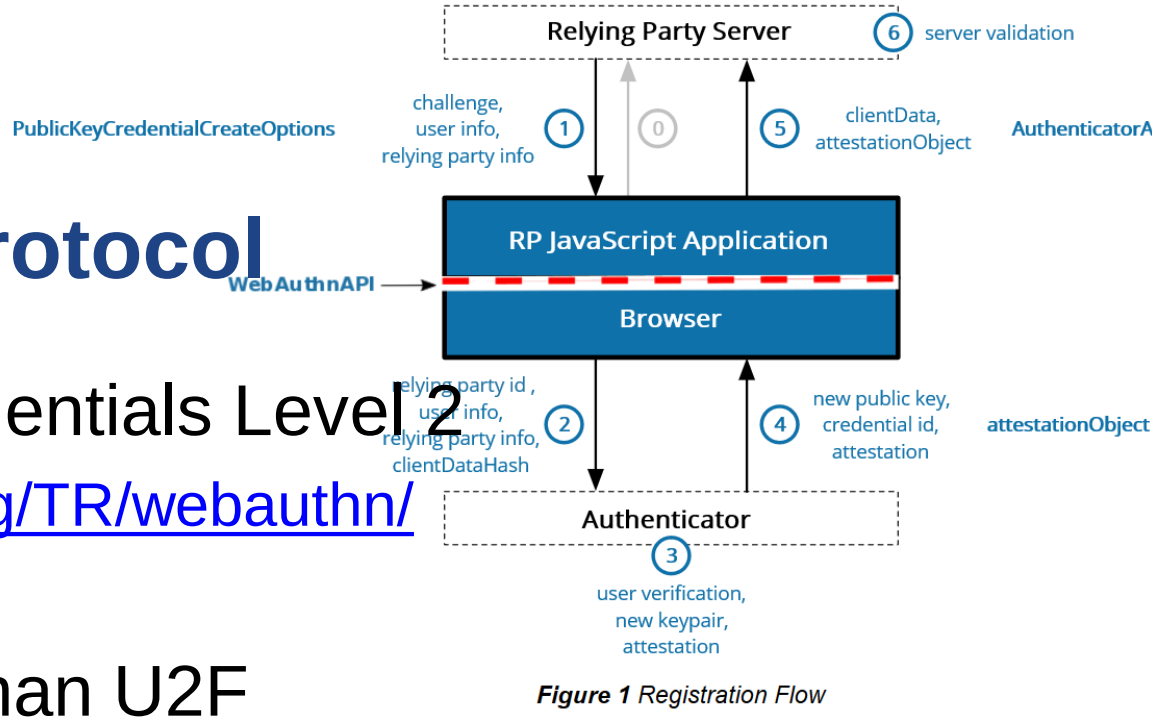


Figure 1 *Registration Flow*

- An API for accessing Public Key Credentials Level 2
  - Official documentation: https://www.w3.org/TR/webauthn/
  - (Level means version here ☺)
- Similar, but more complex standard than U2F
- Client to Authenticator Protocol (CTAP)
  - protocol for communication between browser and token (authenticator)
  - USB, NFC, Bluetooth
    - CTAP 2.2 adds support for the hybrid transport (FIDO Cross-Device Authentication flow, aka Passkeys)
- Explanation, demo page https://webauthn.guide/#about-webauthn

Missing piece? “Passkeys”

- ✓ Authentication on PC with FIDO2 token
- ✓ Authentication on mobile phone (with or without token)
- ? Authentication on PC without FIDO2 token?
- Idea of “passkeys” (multi-device FIDO credentials)
  1. WebAuthn (“U2F”) protocol used for base authentication (private keys needed)
  2. Mobile phone is used instead of FIDO hardware token
  3. Connection between PC and mobile phone done using Bluetooth LE (BLE)
- Now supported natively by Apple (Keychain), Google (Password Manager) and Microsoft (Hello)
  - Cross-compatible between vendors – e.g., Windows Hello together with iPhone
  - https://media.fidoalliance.org/wp-content/uploads/2022/03/How-FIDO-Addresses-a-Full-Range-of-Use-Cases-March24.pdf
  - https://passkeys.dev/docs/reference/specs/

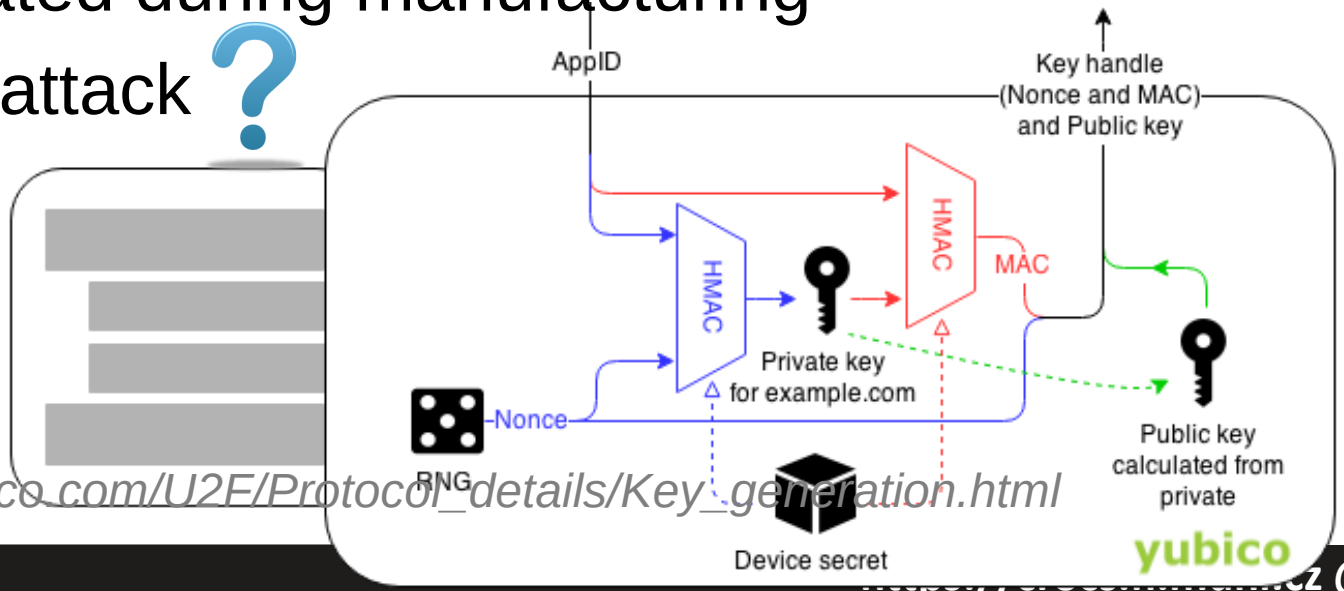
FIDO U2F devices

- Why have button? Is missing display problem?
- Existing past attack: direct WebUSB API in Chrome
  - Malware bypass legitimate U2F API checking the URL, changed URL is send from malicious page, https://www.wired.com/story/chrome-yubikey-phishing-webusb/
  - WebUSB was disabled for certain classes of USB devices as a result
- Well known is Yubikey, but open-source hardware and/or software-only implementations also possible
  - https://github.com/drduh/YubiKey-Guide
  - https://github.com/conorpp/u2f-zero, https://github.com/solokeys/solo

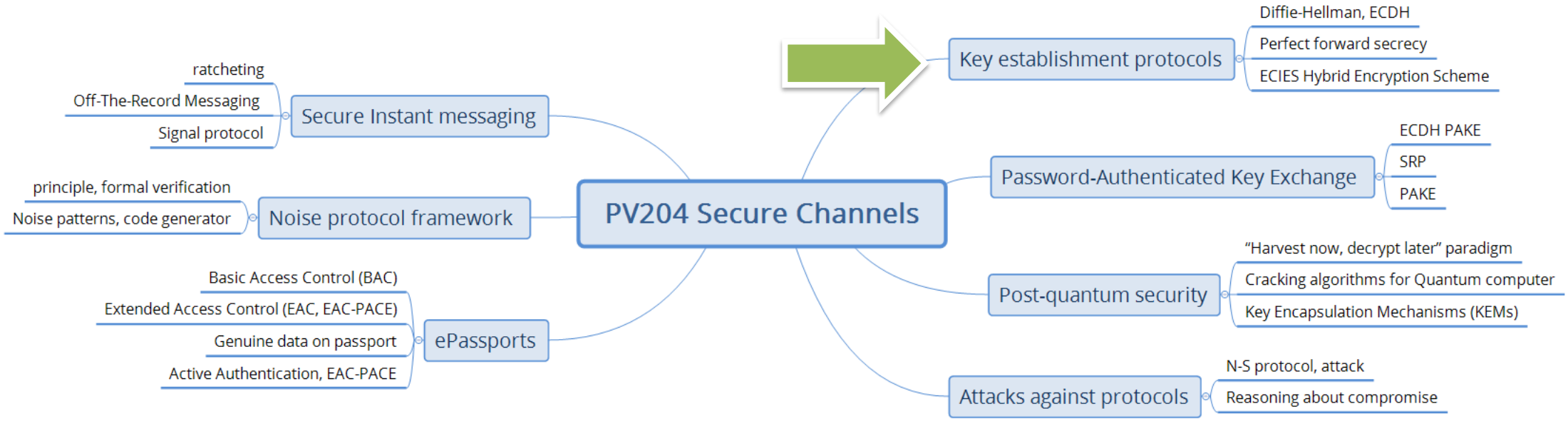
- List of FIDO devices
  - https://opotonniee.github.io/fido-mds-explorer/

Always dig for implementation details

- How are ECC keys generated and stored?
- Yubikey saves storage memory by deriving ECC private keys from master secret instead of randomly generating new one
  - Possible as the ECC private key is random value
- Device secret generated during manufacturing
- What is the possible attack ?

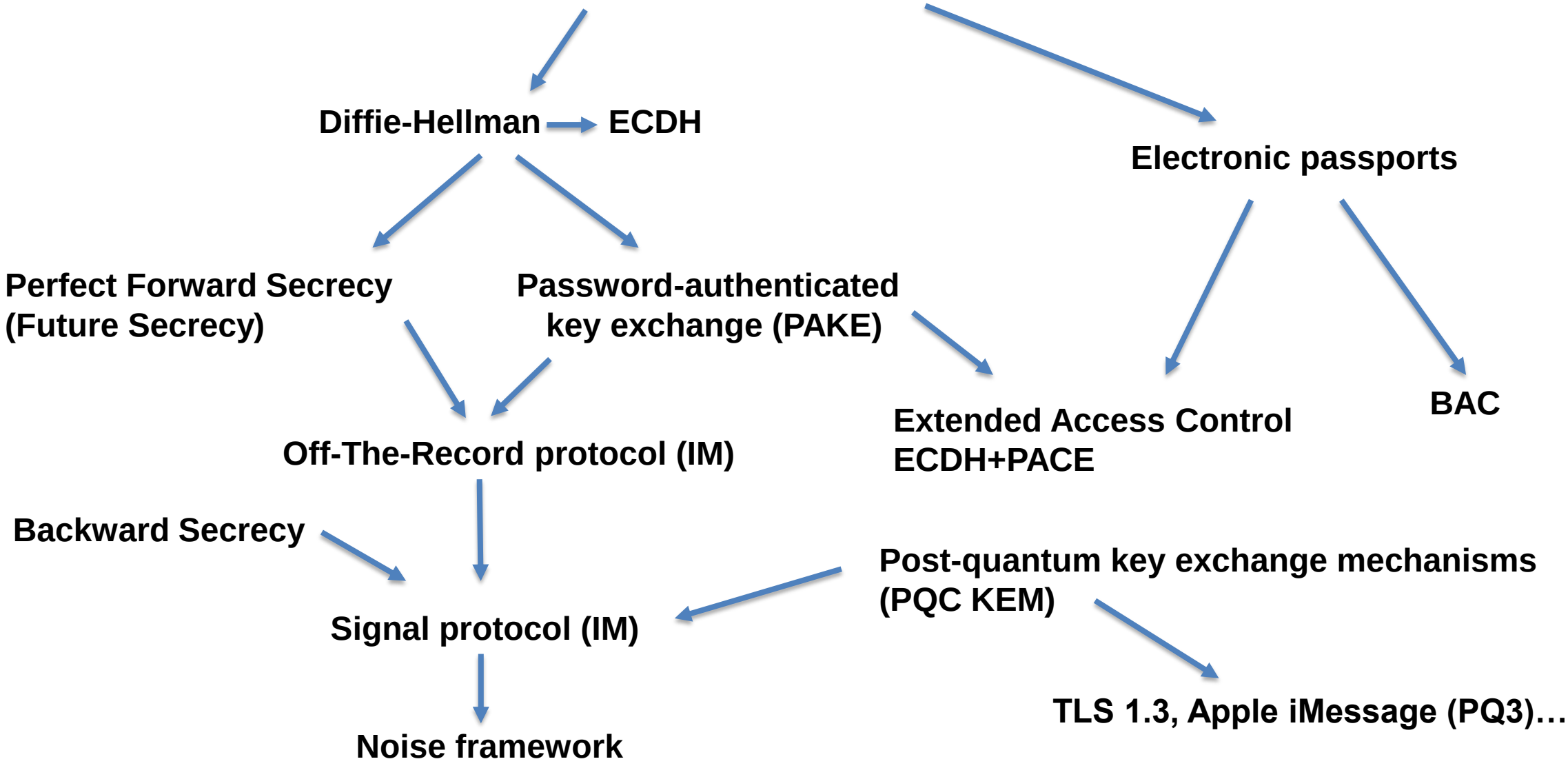


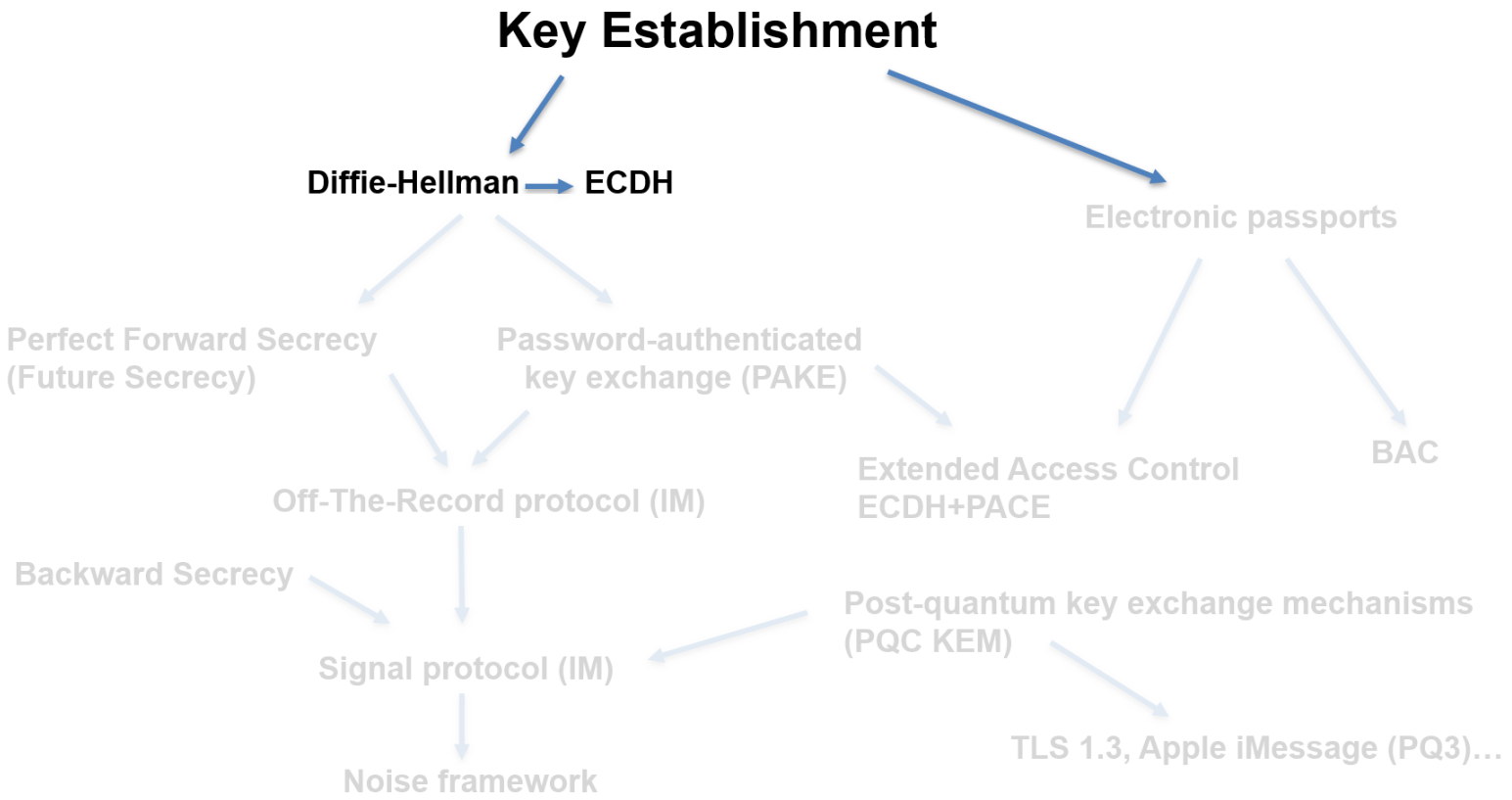
https://developers.yubico.com/U2F/Protocol_details/Key_generation.html



This and next lecture

Key Establishment





KEY ESTABLISHMENT

Methods for key establishment

1. Derive from pre-shared secret (PBKDF2)
2. Establish with help of trusted party (Kerberos, PKI)
3. Establish over insecure channel (Diffie-Hellman)
4. Establish over other (secure) channel (code book)
5. Establish over non-eavesdropable channel (BB84, Quantum effects cryptography)
6. Establish using algorithms resistant to (future) quantum computers (KEMs, Kyber, Post-quantum cryptography)
7. …

Methods for key confirmation

- Goal: ensure that parties use same key value(s)
- Implicit confirmation by use of valid key
  - E.g., MAC by session key on future message is valid
- Explicit confirmation by challenge-response
  - Dedicated steps in protocol

Option	Alice	Bob
1	$R_1 = \text{random}()$	$\text{random}() = R_2$
	$E_{K'}(R_1) \longrightarrow$	
		$\longleftarrow E_{K'}(R_1, R_2)$
	$E_{K'}(R_2) \longrightarrow$	
2	$H(H(K')) \longrightarrow$	
		$\longleftarrow H(K')$

http://www.themccallums.org/nathaniel/2014/10/27/authenticated-key-exchange-with-speke-or-dh-eke/

Diffie-Hellman key exchange

Which part ensures:
Key establishment
Key confirmation
Authentication

Diffie-Hellman Key Exchange

Step	Alice	Bob
1	Parameters: p, g	
2	$A = \text{random}()$ $a = g^A \pmod p$	$\text{random}() = B$ $g^B \pmod p = b$
3	$a \longrightarrow$ $\longleftarrow b$	
4	$K = g^{BA} \pmod p = b^A \pmod p$	$a^B \pmod p = g^{AB} \pmod p = K$
5	$\longleftarrow E_K(data) \longrightarrow$	

Cyclic group with large order, generator g, large prime p

http://www.themccallums.org/nathaniel/2014/10/27/authenticated-key-exchange-with-speke-or-dh-eke/

Diffie-Hellman in practice

- Be aware of particular **p** and **g**
  - If **g** is widely used, then precomputation is possible for lengths up to 1024b
    - "Logjam" attack, [CCS'15]
    - Huge precomputation effort, but feasible for large national agency
    - Certain combination of g and p => fast discrete log to obtain A
  - If p is really prime and g has larger order (Indiscrete logs, [NDSS17])
- Variant of DH based on elliptic curves used (ECDH)
  - ECDH is preferred algorithm for TLS, ePassport…
  - ECDH is algorithm of choice for secure IM (Signal)

DH based on elliptic curves used (ECDH)

EC curve options:
- Edwards curves (e.g., Ed25519)
- NIST FIPS curves (e.g., NIST P-256)
- … many options, see https://safecurves.cr.yp.to/

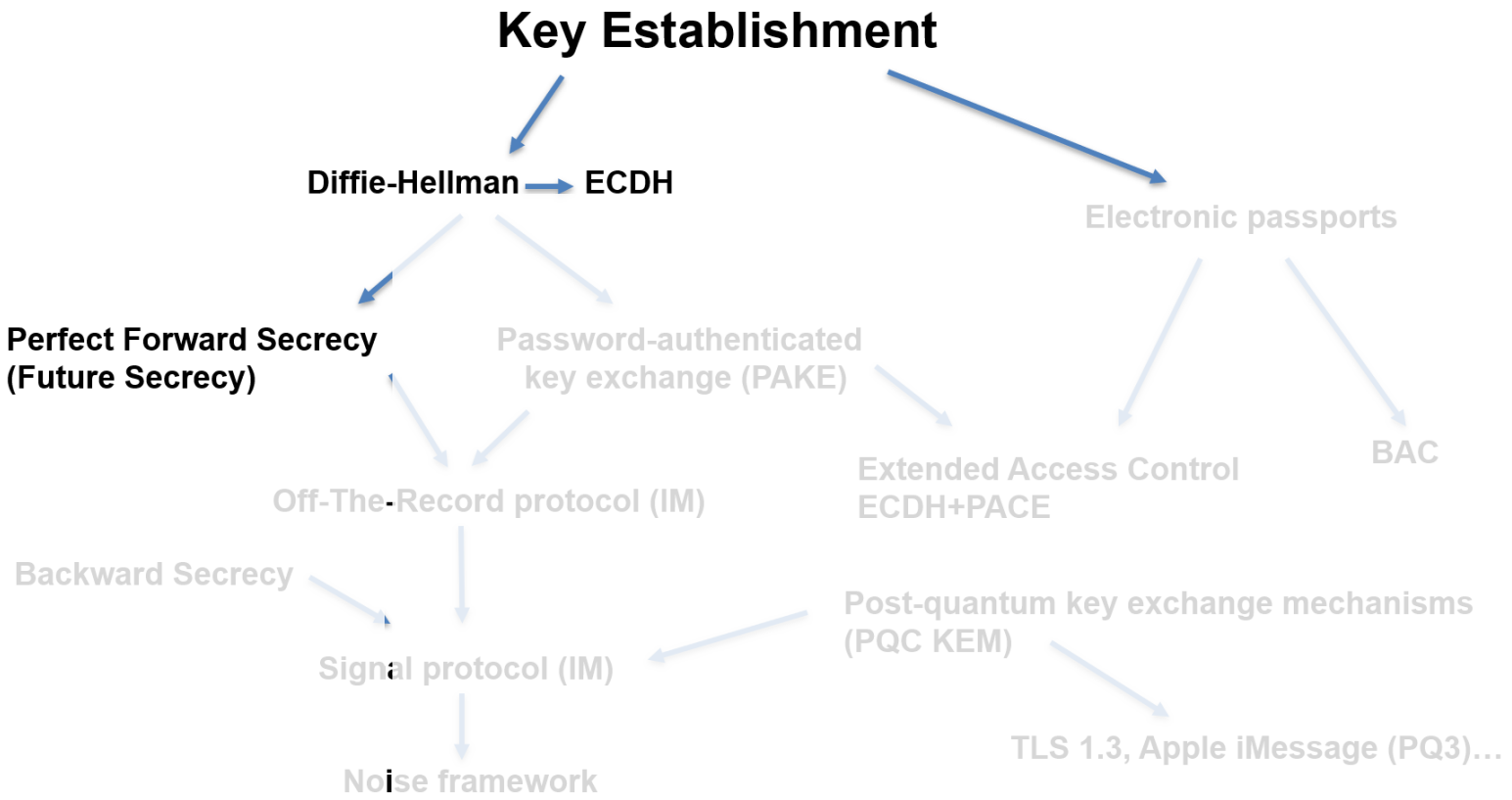
Diffie-Hellman Key Exchange

Step	Alice	Bob
1	Parameters: EC curve, G (base point)	
2	A = random() a = A x G (scalar multiplication)	random() = B B x G = b
3	$a \longrightarrow$ $\longleftarrow b$	
4	K = A x B x G = A x b	B x a = A x B x G = K
5	$\longleftarrow E_K(data) \longrightarrow$	

http://www.themccallums.org/nathaniel/2014/10/27/authenticated-key-exchange-with-speke-or-dh-eke/

Diffie-Hellman in practice

- K is not used directly, but K' = KDF(K) is used
  1. Original K may have weak bits (biased value not uniformly distributed)
  2. Multiple keys may be required (e.g., K_{ENC}, K_{MAC}) => key derivation
- Is vulnerable to man-in-the-middle attack (MitM)
  - Attacker runs separate DH with A and B simultaneously
  - (Unless values a and b are authenticated)
- DH can be used as basis for *Data encryption* (session key, one-shot)
- DH can be used as basis for *Forward/Backward secrecy*
- DH can be used as basis for *Password-Authenticated Key Exchange*



PERFECT FORWARD SECRECY

Forward secrecy - motivation

- Assume that session keys are exchanged using long-term secrets
  1. Pre-distributed symmetric cryptography keys (SCP'02)
  2. Public key cryptography (PGP, TLS_RSA_...)
- What if long-term secret is compromised?
  I. All future transmissions can be read
  II. Attacker can impersonate user in future sessions
  III. All previous transmissions can be compromised (if traffic was captured)
- Can III. be prevented? (Forward secrecy)
- Can I. be prevented? (Backward secrecy, "healing")

Must not have past keys

Must not derive future keys deterministically

Forward/backward secrecy – how to

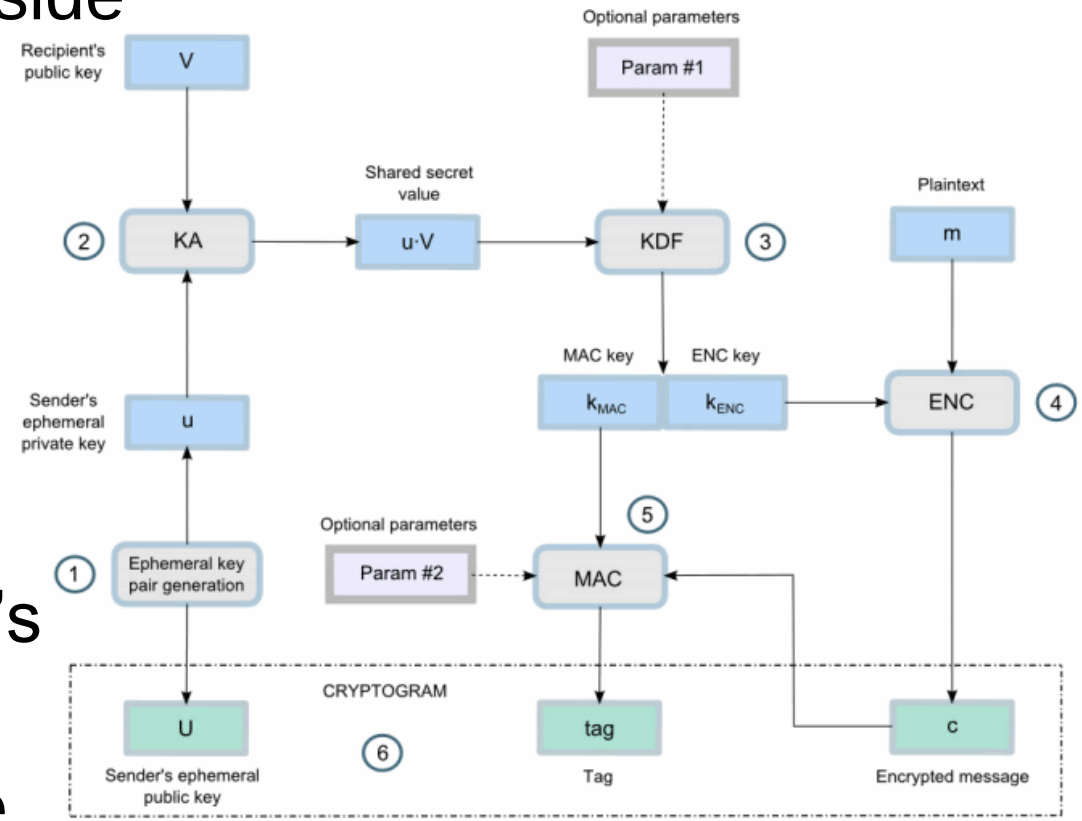
- (Perfect) Forward Secrecy
  – Compromise of long-term keys does not compromise past session keys
- Solution: ephemeral key pair (DH/ECDH/RSA/…)
  1. Fresh keypair generated for every new session
  2. Ephemeral public key used to exchange session key
  3. Ephemeral private key is destroyed after key exchange
     - Captured encrypted transmission cannot be decrypted later
- Long-term key is used only to authenticate ephemeral public key to prevent MitM
  – E.g., MAC over DH share

Use of forward secrecy: examples

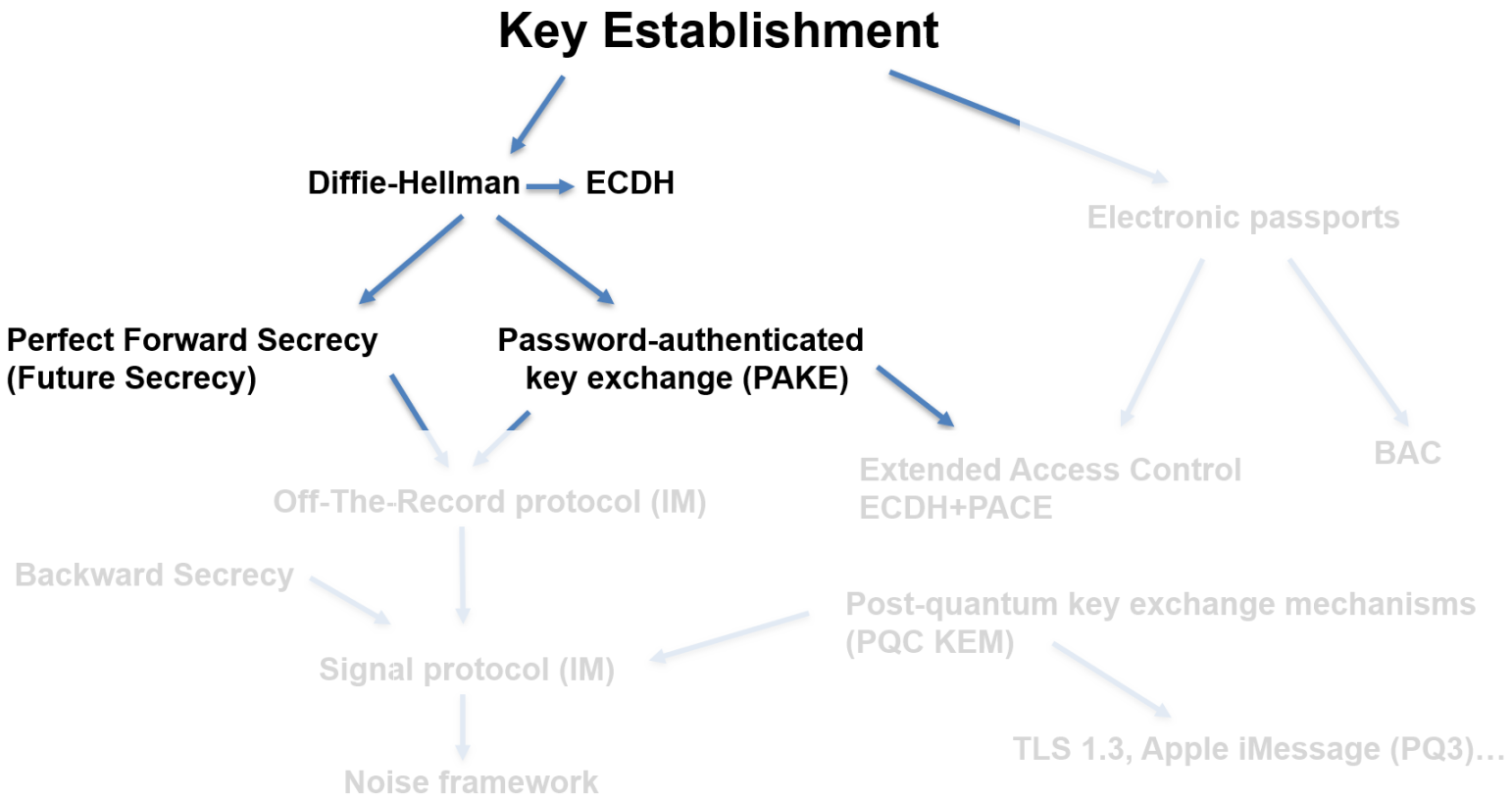
- HTTPS / TLS
  - TLS1.2: ECDHE-ECDSA, ECDHE-RSA…
  - TLS1.3: TLS_ECDHE_ECDSA_WITH_xxx…
- SSH (RFC 4251)
- PAKE protocols: EKE, SPEKE, SRP…
- Off-the-Record Messaging (OTR) protocol (2004)
- Signal protocol (2015)
- Noise protocol framework (2017)

ECDH for data encryption: ECIES Hybrid Encryption Scheme

1. New ephemeral ECDH key on sender side
2. ECDH with recipient pub key
3. Key derivation for symmetric key(s)
4. Data Encryption (ENC)
5. Data Authentication (MAC)
6. Message blob (Cryptogram)
7. Recipient performs ECDH with sender's eph. pub key, recovers k_{ENC},k_{MAC}
8. Recipient decrypts and verify message



(ECIES does not provide forward secrecy if recipient's key is long-term). Why?



PASSWORD-AUTHENTICATED KEY EXCHANGE (PAKE)

PAKE protocols - motivation

- Diffie-Hellman can be used for key establishment
  - Authentication ca be added via pre-shared (long-term) key
- But why not directly derive session keys from pre-shared instead of running DH?
  1. Compromise of pre-shared key => compromise of all data transmissions (including past) => no forward secrecy
  2. Pre-shared key can have low entropy (password / PIN) => attacker can brute-force
- Password-Authenticated Key Exchange (PAKE)
  - Sometimes called “key escalation protocols”

PAKE protocols - principle

- Goal: prevent MitM and offline brute-force attack

1. Generate asymmetric keypair for every session
   - Both RSA and DH possible, but DH provides better performance in keypair generation
2. Authenticate public key by (potentially weak) shared secret (e.g., password or even PIN)
   - Must limit number of failed authentication requests!
3. Exchange/establish session keys for symmetric key cryptography using authenticated public key

Diffie-Hellman Encrypted Key Exchange [PAKE]

Step	Alice	Bob
1	Shared Secret: $S = H(password)$	
2	Parameters: p, g	
3	$A = \text{random}()$ $a = g^A \pmod p$	$\text{random}() = B$ $g^B \pmod p = b$
4a	$E_S(a) \longrightarrow$ $\longleftarrow E_S(b)$	
4b	$a \longrightarrow$ $\longleftarrow E_S(b)$	
4c	$E_S(a) \longrightarrow$ $\longleftarrow b$	
5	$K = g^{BA} \pmod p = b^A \pmod p$	$a^B \pmod p = g^{AB} \pmod p = K$
6	$\longleftarrow E_K(data) \longrightarrow$	

Various options a,b,c available

To protect against offline bruteforce attack against the password used, an attacker must not be able to tell if the decrypted a' is valid (any structure of a' can reveal successful decryption)

Simple Password Exponential Key Exchange (SPEKE)

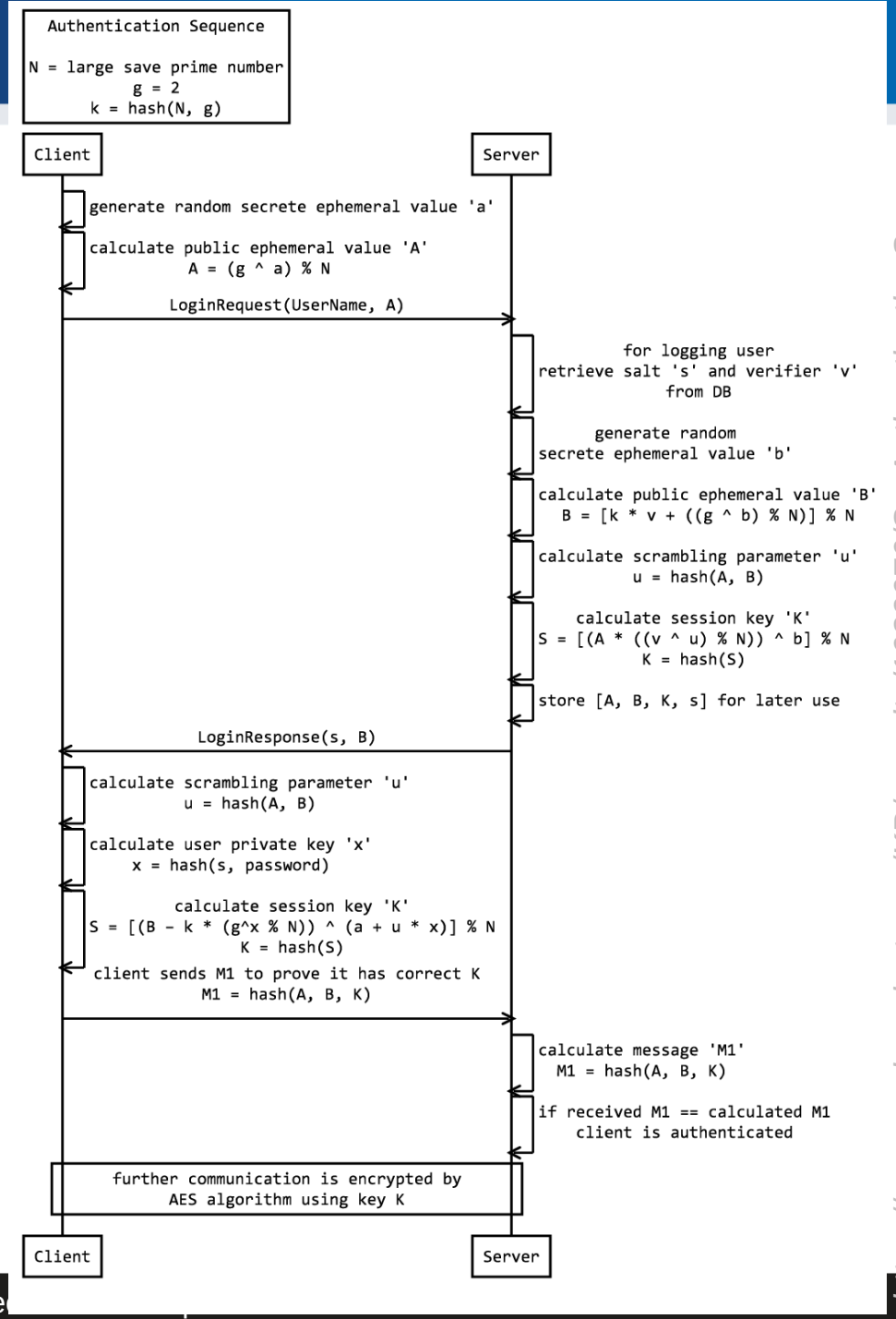
Simple Password Exponential Key Exchange

Step	Alice	Bob
1	Parameter: p	
2	$G = H(password)^2$	$H(password)^2 = G$
3	$A = \text{random}()$ $a = G^A \pmod p$	$\text{random}() = B$ $G^B \pmod p = b$
4	$a \longrightarrow$ $\longleftarrow b$	
5	$K = G^{BA} \pmod p = b^A \pmod p$	$a^B \pmod p = G^{AB} \pmod p = K$
6	$\longleftarrow E_K(data) \longrightarrow$	

http://www.themccallums.org/nathaniel/2014/10/27/authenticated-key-exchange-with-speke-or-dh-eke/

Secure Remote Password protocol (SRP), [aPAKE]

- Earlier Password-Authenticated Key Exchange protocols (PAKEs) were patented: EKE, SPEKE… (patents expired till 2017)
- Secure Remote Password protocol (SRP) 1998
  - Designed to work around existing patents
  - Royalty free, open license (Standford university), basis for multiple RFCs
  - Several revisions since 1998 (currently 6a)
  - Originally with DH, variants with ECDH exist
  - Widely used, support in common cryptographic libraries
- Apple uses SRP extensively in its iCloud Key Vault
- Asymmetric Password Authenticated Key Exchange (aPAKE)

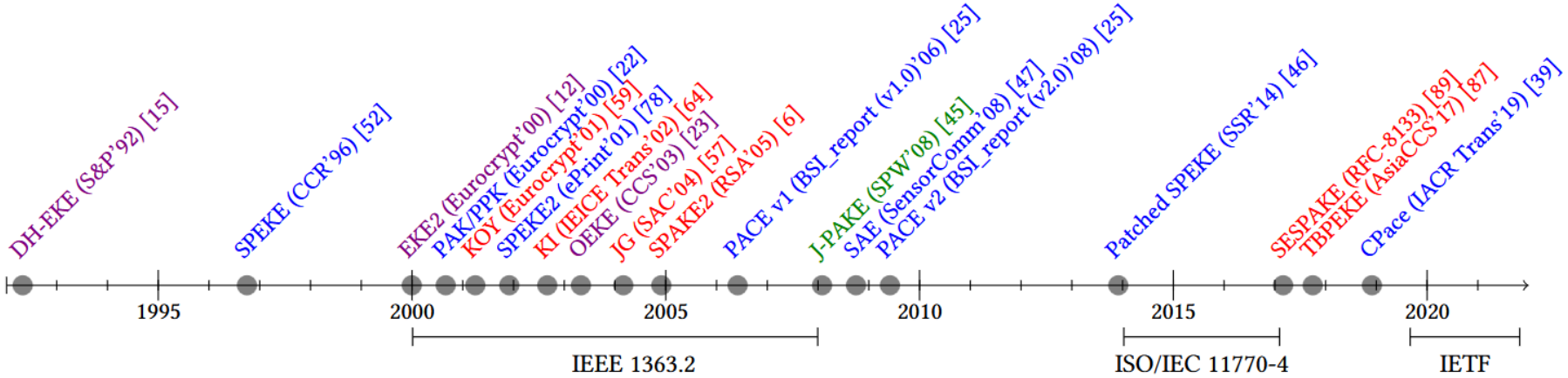


https://www.codeproject.com/KB/security/1082676/SrpAuthenticationSequence.png

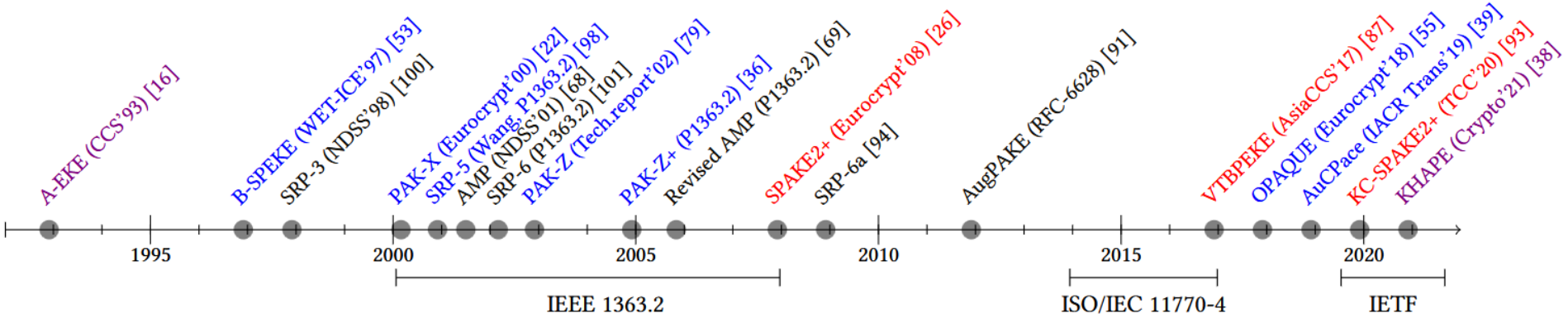
SRP is unnecessarily complex (to work around existing patented protocols)!

https://eprint.iacr.org/2021/1492.pdf

Long history of PAKE improvements



(a) Balanced PAKE = both parties share same secret (e.g., password)



(b) Augmented PAKE = server compromise resistance (server stores only "hash(pass))"

PAKEs evolution

1. Only password
2. “PAKE” protocols (balanced PAKE)
3. “aPAKE” protocols (augmented PAKE)
4. Strong aPAKE (“SaPAKE”)

Properties

- Compromised if server hack
- Prevent MitM offline cracking, vulnerable to server hack
- Like PAKE, but using salted hash instead of password, salt-specific precomputation possible
- Prevent offline cracking and pre-computation attack (using zero-knowledge proofs)

https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/

Current state of the art SaPAKE

- OPAQUE protocol (Eurocrypt 2018)
  - https://eprint.iacr.org/2018/163.pdf
  - https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/
- Prediction about future PAKE uptake
  - https://emilymstark.com/2020/07/30/should-web-apps-use-pakes.html

Practical deployment of PAKE protocols

- Challenges in adoption (since 1992)
  - Early protocols were patented
  - Lack of standardization (IETF is working on future RFC)
  - Performance concerns (involves asymmetric cryptographic operation => slow in limited clients / highly-used servers), lack of awareness and production-ready implementations
  - Now seems to be finally changing!
- Notable practical deployments of PAKE protocols
  - Secure Remote Password (SRP-6a) protocol
    - Apple's iCloud security codes protection
    - 1Password, ProtonMail use SRP for client to server authentication
  - Wi-Fi Protected Setup (WPS) RFC7664
  - Apple car unlock protocol: SPAKE2+ https://support.apple.com/guide/security/car-key-security-secf64471c16/web

SoK: Password-Authenticated Key Exchange – Theory, Practice, Standardization and Real-World Lessons (2022)

- https://eprint.iacr.org/2021/1492.pdf
- Real-world use cases!

Use case	Methods	Example	Type	Property: No dependence on a PKI	Property: Prevents external attacks	Property: Prevents insider attacks
Credential recovery	SRP-6a	iCloud	Preventive	◐	●	○
	PKI+pwd	Google sync	Preventive	○	●	○
Device pairing	PACE	E-passport	Preventive	●	●	–
	Dragonfly	WPA3	Preventive	●	◐	–
	Passkey	Bluetooth	Preventive	●	○	–
	Num com†	Bluetooth	Detective	●	●	–
E2E secure channel	J-PAKE	Thread	Preventive	●	●	●
	EC-SPEKE	BBM	Preventive	●	◐	●
	PKI+pwd	TLS 1.3	Preventive	○	●	●
	Signal†	WhatsApp	Detective	●	●	●
	ZRTP†	VoIP	Detective	●	●	●

● (provides property); ◐ (partially provides); ○ (does not provide); – (not applicable). †Requires manual security-check by end-users.

Table 4: Use case properties: PAKE, non-PAKE

C1 **Password used as encryption key**. This class includes using a password as encryption key, and typically must assume an ideal cipher. Examples: EKE [15], EKE2 [12], OEKE [23], A-EKE [16], KHAPE [38].

C2 **Password-derived generator**. A protocol group generator is derived from a password. Examples: SPEKE [52], SPEKE2 [80], Patched SPEKE [44] and B-SPEKE [53], PAK [79], SAE (Dragonfly) [47], PACE [30], SRP-5 [98], CPace/AuCPace [39], OPAQUE [55].

C3 **Trusted setup**. The protocol relies on a trusted setup, which defines two (or more) generators whose discrete logarithm relationship must be unknown. Examples: KOY [59], KI [64], JG [57], SPAKE2 [6], SESPAKE [89], TBPEKE/VTBPEKE [87], KC-SPAKE2+ [93].

C4 **Secure two-party computation**. Here PAKE is viewed as a two-party secure computation problem on an equality function; use of a non-interactive zero-knowledge proof (ZKP) aims to check that parties follow a specification honestly. Example: J-PAKE [45].

C5 **Password-derived exponent**. In this class, a password is used to derive g^w as a verifier in a type of Diffie-Hellman key exchange. Examples: SRP-3 [100], SRP-6 [101], AMP [68], revised AMP [69], AugPAKE [92].

POST-QUANTUM KEY EXCHANGE MECHANISMS (PQC KEM)

Impact of quantum computer

- Quantum computer uses different computational paradigm
  - Small ones already build
- Sufficiently large quantum computer needed to attack cryptographic algorithms
  - Estimated number of required logical qubits for factorization of RSA 2048b is ~4000 logical qubits; roughly 20 million physical qubits https://arxiv.org/abs/1905.09749 ; https://sam-jaques.appspot.com/quantum_landscape_2024
- Physical vs. logical qubits
  - Google's Willow chip (12/2024) has 105 physical qubits https://blog.google/technology/research/google-willow-quantum-chip/
    - Many physical qubits (~100-1000) required to form one logical qubit (to suppress errors)
    - Longer the algorithm runs => more error cumulates => more physical qubits are required for a single logical qubit
    - Willow chip was used to create only a single logical qubit

Quantum algs. for breaking classical crypto algorithms

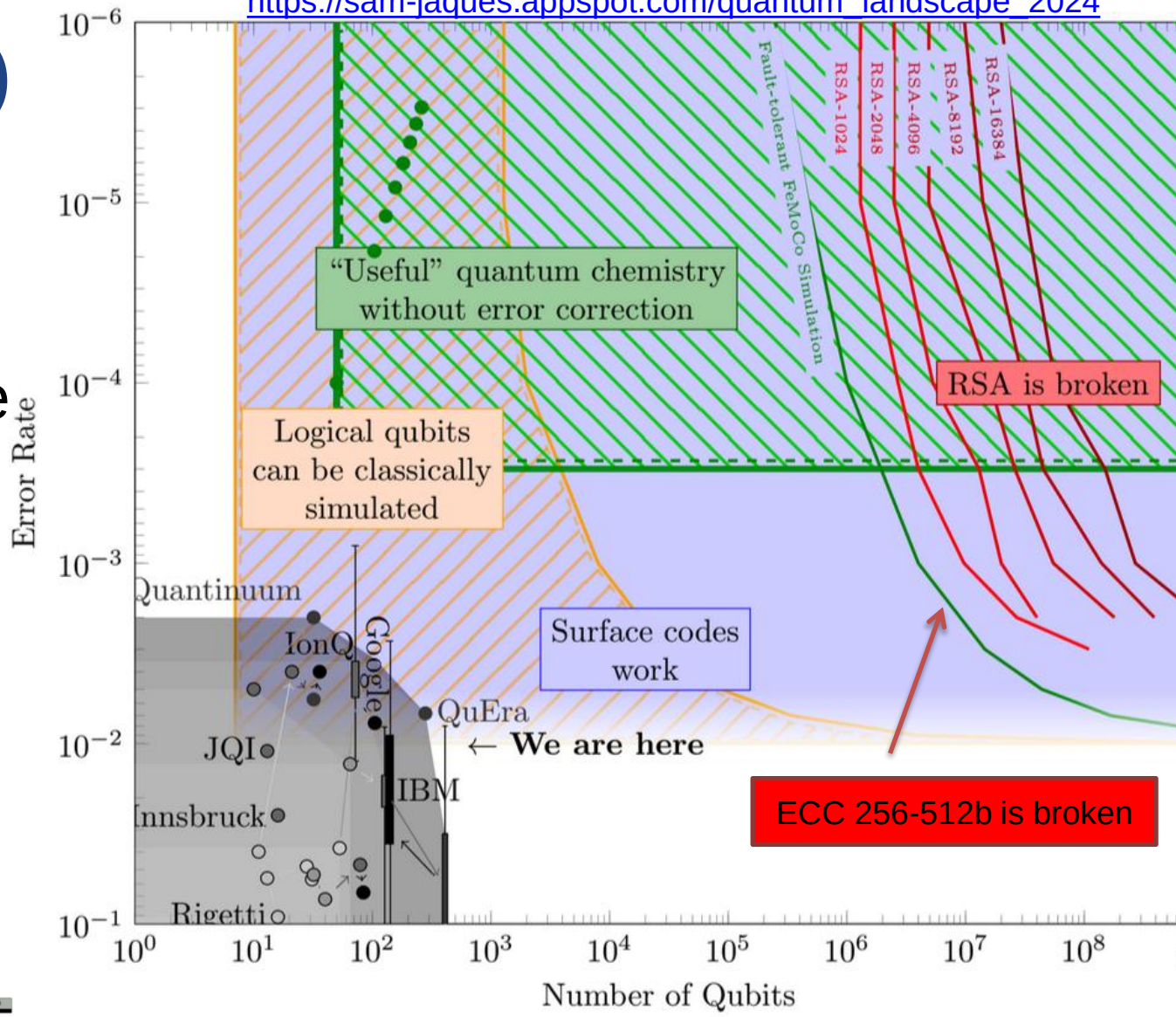
- Asymmetric algorithms: RSA (factorization), (EC-)DSA (discrete log.)
  - Schor's algorithm (1997) => complete break of classical asymmetric algorithms with currently used key lengths
  - https://epubs.siam.org/doi/10.1137/S0097539795293172
- Symmetric algorithms: block ciphers (AES), hash functions (SHA2/3)
  - The attack typically requires brute-force search over key/message space
  - Grover's algorithm (1996) for efficient brute-force => security degrades to half of bit length (e.g., AES-256 degrades to 128 bits of security)
  - https://dl.acm.org/doi/10.1145/237814.237866
- No proof that Schor / Grover algorithms are the best possible!
  - but best known for a long time

Quantum landscape (2024)

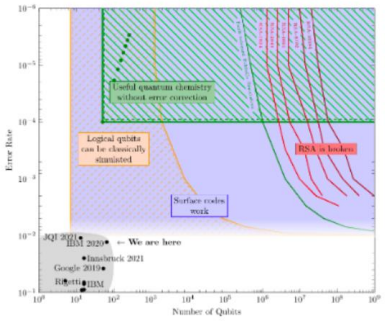
"Harvest now, decrypt later"

- Surface codes correct errors
- Classic computers can simulate up to ~40 logical qubits
- Mind logarithmic axes!

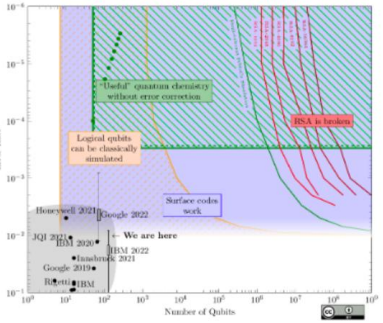
https://sam-jaques.appspot.com/quantum_landscape
https://sam-jaques.appspot.com/quantum_landscape_2024



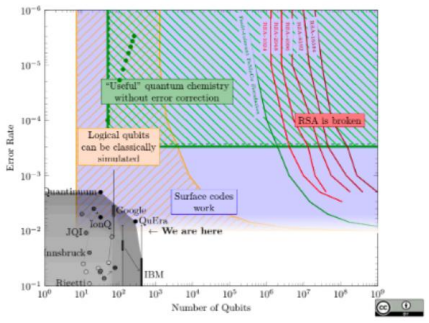
2021



2022



2023



Key Encapsulation Mechanisms (KEMs), Signatures

- Ongoing research, NIST PQC competition
- Key Encapsulation Mechanisms
  - How to establish session key for symmetric alg. (e.g., AES) for transfer
  - Post-quantum analogues to ECDH
    - NIST ML-KEM (also known as CRYSTALS-Kyber)
    - SIDH, CSIDH (elliptic curve isogenies)
- Digital signatures
  - How to sign message (PQC analogue to RSA/ECDSA)
  - CRYSTALS-Dilithium…
  - Range of possible tradeoffs between signature/verification speed and size

Keys and messages are longer for PQC

- Key/message lengths
  - RSA-2048b: 256 bytes public key & private key & ciphertext/signature
  - ECC-256b: 32 bytes public key & private key, 32 bytes ciphertext/signature
  - Kyber-768: 1184 bytes public key, 2400B private key, 1088B ciphertext
- Performance
  - NIST ML-KEM (Kyber) is very fast
    - Even faster than ECC and RSA
  - Some PQC schemes may be slower
  - Bigger downside are lengths, not speed

Algorithm	Security Level (Classical)	Public Key Size	Ciphertext Size	Secret Key Size	Encapsulation Time	Decapsulation Time
Kyber-768 (PQC)	~192-bit	1,184 bytes	1,088 bytes	2,400 bytes	~70µs	~100µs
RSA-2048	~112-bit	256 bytes	256 bytes	2,048 bytes	~1ms	~6ms
ECC (P-256)	~128-bit	32 bytes	32 bytes	32 bytes	~200µs	~300µs
AES-128	~128-bit	- (Symmetric)	16 bytes (Block Size)	16 bytes (Key)	~10ns per block	~10ns per block

Generated by ChatGPT, verified via https://asecuritysite.com/pqc/kem768

Practical adoption of post-quantum cryptography

- PQC used typically in the hybrid mode (Hybrid-PQC)
  - Both classical and PQC algorithm used together, e.g., X25519_Kyber768
  - Classical security still guarantied even if flaw in (newer) PQC algorithm or its implementation is found
- Practical adoption
  - TLS1.3 support (X25519_Kyber768, key exchange during handshake)
    - ~13% of TLS connections secured by PQC KEM (12/2024)
  - Signal IM (PQXDH), Apple iMessage (PQ3) – passive quantum adversary only
- Practical considerations https://blog.cloudflare.com/pq-2024/
  - Engineering challenges for PQC transition

Conclusions

- Almost all security protocols require some keys
  - Key establishment and key management is crucial
- Resiliency against compromise of long-term secrets is important (forward secrecy)
- Strong session keys can be authenticated by weak passwords (PAKE protocols)
- Quantum Computer breaks classical asymmetric crypto algs. (if build large enough)
  - Post-quantum cryptography algorithms now readily available (KEMs, Signatures – CRYSTALS-Kyber, CRYSTALS-Dillitium)
  - “Harvest now, decrypt later” paradigm
- Mandatory reading
  - The state of the post-quantum Internet, https://blog.cloudflare.com/pq-2024/