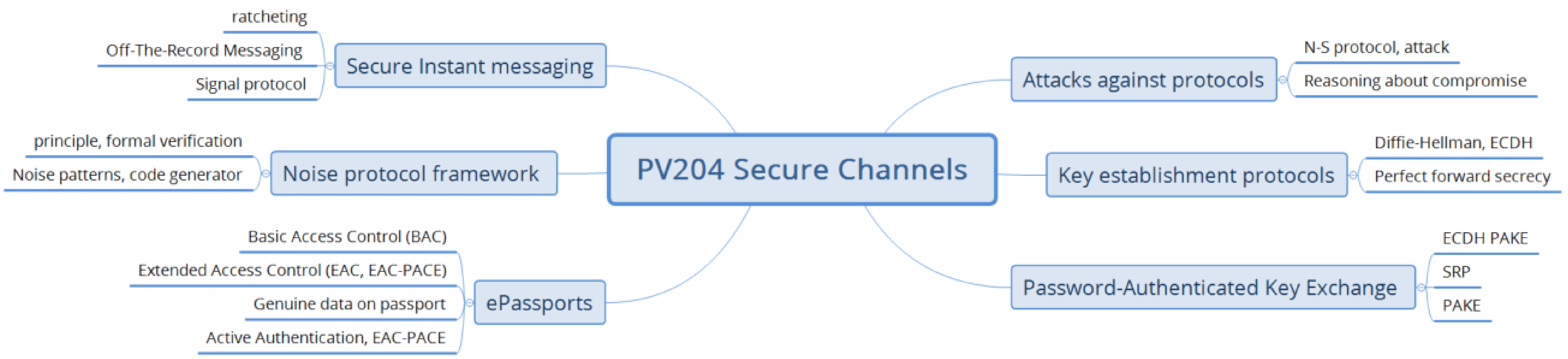
PV204 Security technologies

Secure channel protocols: From basic key establishment via Signal protocol to Noise framework

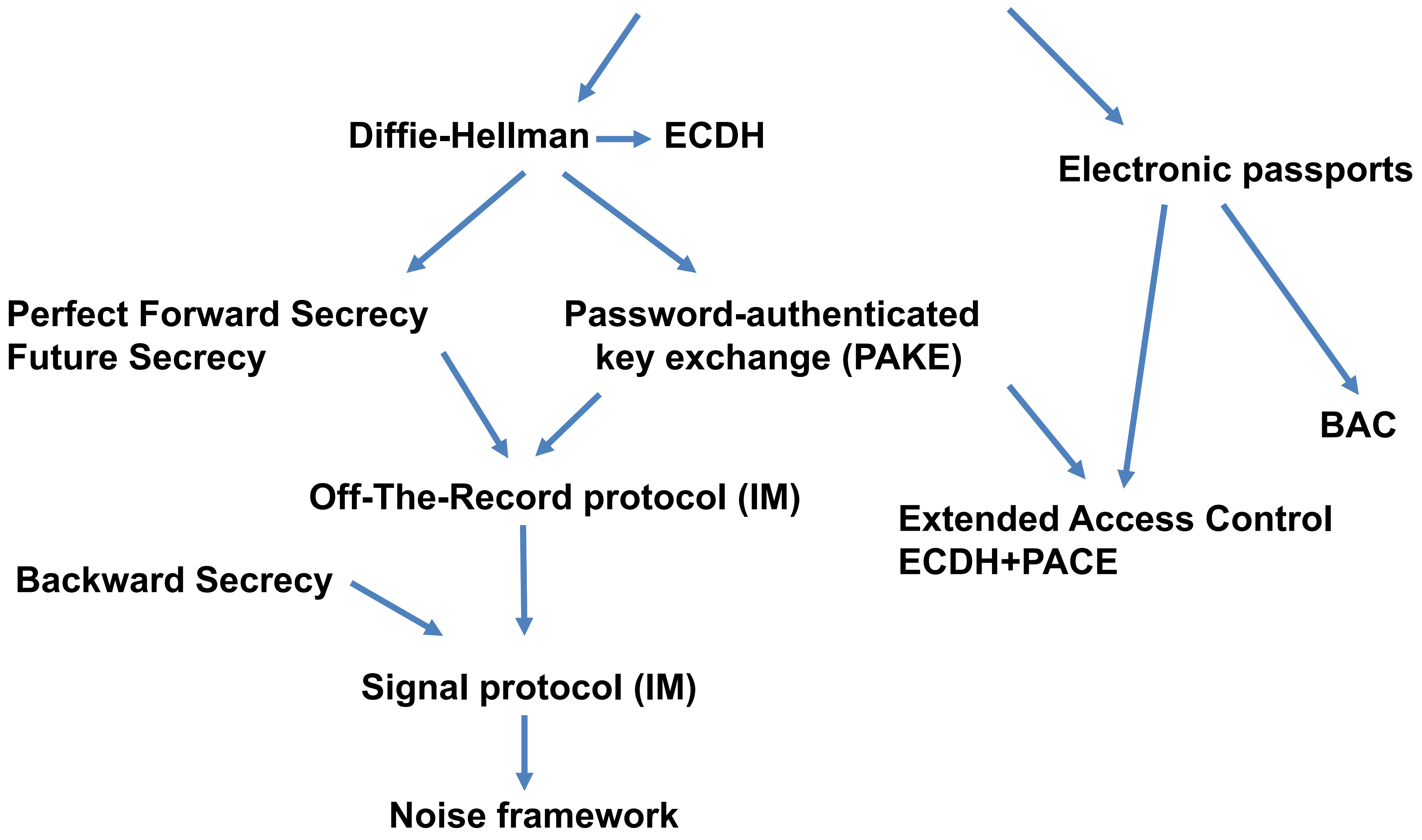
Petr Švenda *svenda@fi.muni.cz* *@rngsec*
Centre for Research on Cryptography and Security, Masaryk University

Please provide any corrections and comments here (thank you!):
https://drive.google.com/file/d/1SXnzLqYlucs-96uTx5VeXB2scI5o3lcC/view?usp=sharing

CRoCS
Centre for Research on Cryptography and Security


PV204 Secure Channels
Secure Instant messaging
ratcheting
Off-The-Record Messaging
Signal protocol
Noise protocol framework
principle, formal verification
Noise patterns, code generator
ePassports
Basic Access Control (BAC)
Extended Access Control (EAC, EAC-PACE)
Genuine data on passport
Active Authentication, EAC-PACE
Attacks against protocols
N-S protocol, attack
Reasoning about compromise
Key establishment protocols
Diffie-Hellman, ECDH
Perfect forward secrecy
Password-Authenticated Key Exchange
ECDH PAKE
SRP
PAKE

Key Establishment



SECURITY PROTOCOLS

Security protocols

- Security protocol = composition of cryptoprimitives
- *“Security protocols are three line programs that people still manage to get wrong.” (R. Needham)*

- Many different aspects of security protocols
  - Entity authentication
  - Key agreement, establishment or distribution
  - Data encryption and integrity protection
  - Non-repudiation
  - Secure multi-party computation (SMPC)
  - …

Authentication (AUTH) vs. Key establishment (KE)

- Early literature called protocols used to establish session keys as “authentication protocols”
- Session keys can be established without authentication
  - Example: non-authenticated Diffie-Hellman
- Authentication is also possible without session keys
  - Example: Challenge-response protocol like FIDO U2F

- Common protocol workflow (e.g., TLS):
  1. Authenticate parties
  2. Establish session keys
  3. Use session keys to encrypt and authenticate messages
  - (do it in as few messages as possible)

PROTOCOLS AND ATTACKS

Typical models of adversary

- Adversary controls the communication
  - Between all principals
  - Observe, alter, insert, delay or delete messages
- Adversary can obtain session/long term keys
  - used in previous runs
- Malicious insider
  - adversary is legitimate protocol principal
- Attacker can obtain partial knowledge
  - Secrets compromise, side-channels…
- …

Needham–Schroeder protocol: symmetric

- Basis for Kerberos protocol (AUTH, KE), 1978
  - Two-party protocol (A,B) + trusted server (S)
  - Session key K_{AB} generated by S and distributed to A together with part intended for B
  - Parties A and B are authenticated via S

1. $A \rightarrow S: A, B, N_A$
2. $S \rightarrow A: \{N_A, K_{AB}, B, \{K_{AB}, A\}K_{BS}\}K_{AS}$
3. $A \rightarrow B:$ [redacted]
4. $B \rightarrow A: \{N_B, A\}K_{AB}$
5. $A \rightarrow B: \{N_B - 1\}K_{AB}$

Which part ensures:
Authentication
Key confirmatic
Freshness

?

? Can you spot problem?

N-S symmetric: Problem?

- Vulnerable to replay attack (Denning, Sacco, 1981)
- If an attacker compromised older K_{AB} then
  - $\{K_{AB}, A\}K_{BS}$ can be replayed to B (step 3.)
  - B will not be able to tell if K_{AB} is fresh
  - Attacker will then impersonate A using old (replayed, compromised) key K_{AB}
- Fixed by inclusion of nonce/timestamp $\mathbf{N'_B}$ generated by B (two additional steps before step 1.)
  - Bob can now check freshness of $\{K_{AB}, A, \mathbf{N'_B}\}K_{BS}$

Vhat is required attacker model to perform the attack?

What is required attacker model?

- Able to capture valid communication ($\{K_{AB}, A\}K_{BS}$)
- Able to compromise older K_{AB}
- Actively communicate with B (reply ($\{K_{AB}, A\}K_{BS}$)

But is an assumption of compromise of old key realistic?

How (not) to reason about potential compromise

- NO: all my (many) keys are in secure hardware and therefore I'm secure (no compromise possible)
  – Nothing like perfect security exists

- YES: assume compromise and evaluate impact
  – Where the sensitive keys are
  – How hard is to compromise them
  – What will be the impact of the compromise
  – Can I limit number/exposure of keys? For what price?

What if key is compromised?

- Prevention, detection (is hard), reaction
- Prevention of compromise
  - Limit usage of a key
    - master key $\rightarrow$ session keys (use secure key derivation function)
    - Use PKI instead of many symmetric keys in trusted terminals
  - Limit key availability
    - Erase after use, no/limited copy in memory, trusted element
  - Limited-time usefulness of keys (key update)
    - (Perfect) forward secrecy: messages sent before is secure
- Reaction on compromise
  - stop using key, update and let know (revocation)

Formal verification of protocols

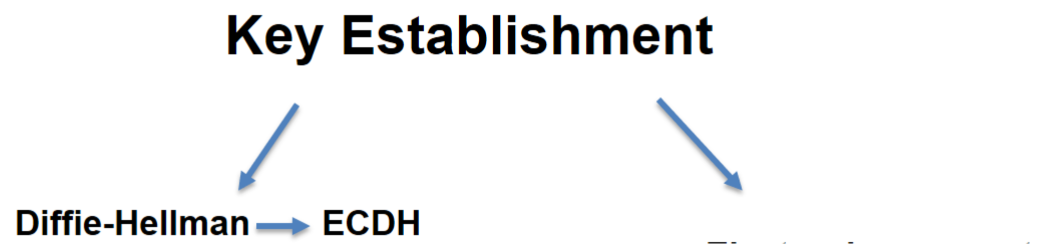
- Negatives
- Specific attacker model
  - Different attacker (e.g., side-channels) => attack possible
- Assumes perfect crypto-primitives
- Sensitive to precise specification
- Hard to express real-world complex protocols
  - Search space too large

- Positives
- Automated process
- Prevents basic and some advanced design flaws
- Favours simple solutions
  - Complexity is enemy of security

? s formal verification panacea?

Proofs by formal verification now considered good practice and actively aimed for (e.g., TLS1.3)

MORALE: PROTOCOL DESIGN IS VERY HARD => USE EXISTING, PROVEN ONES



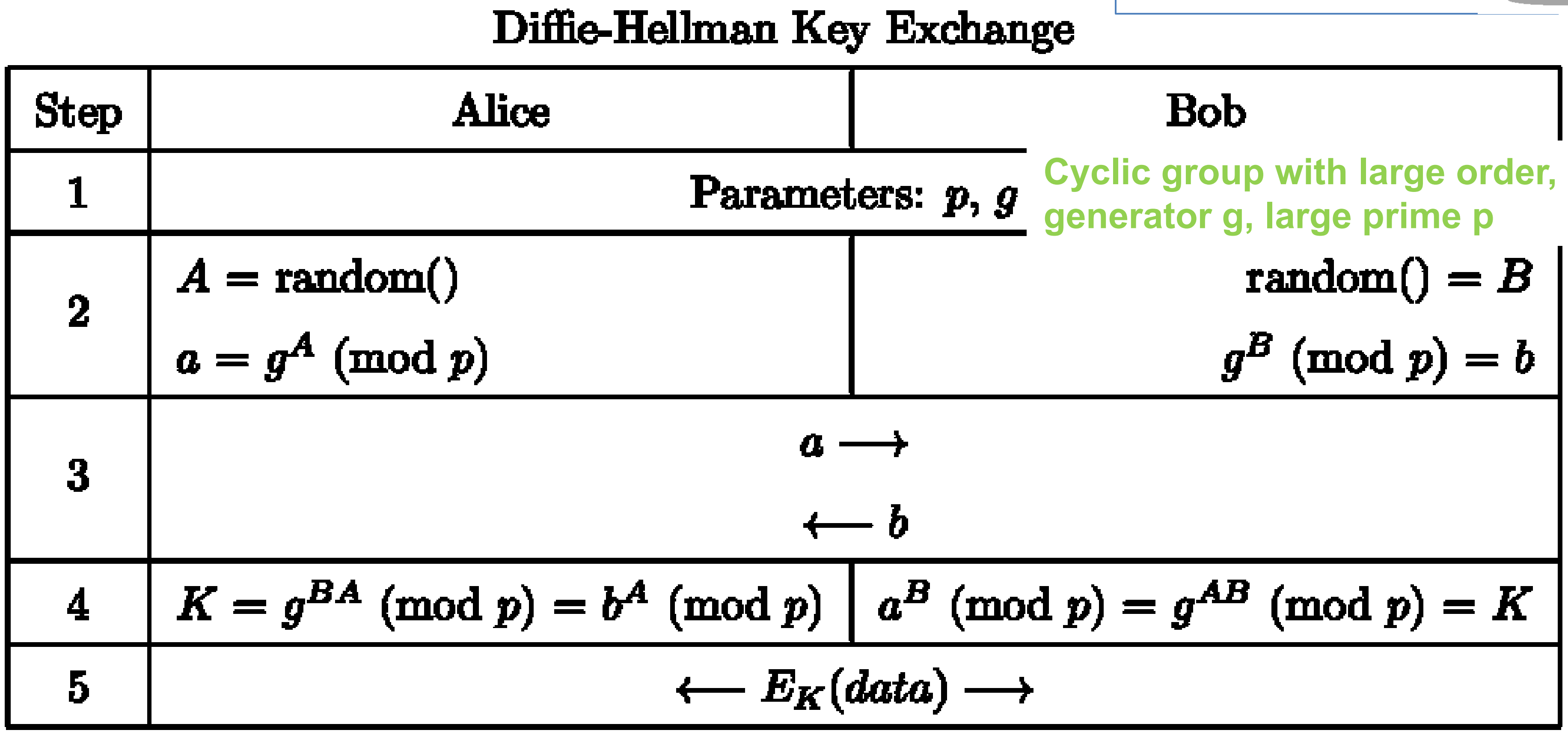
KEY ESTABLISHMENT

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)
6. …

Diffie-Hellman key exchange

Which part ensures:
Key establishmer
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

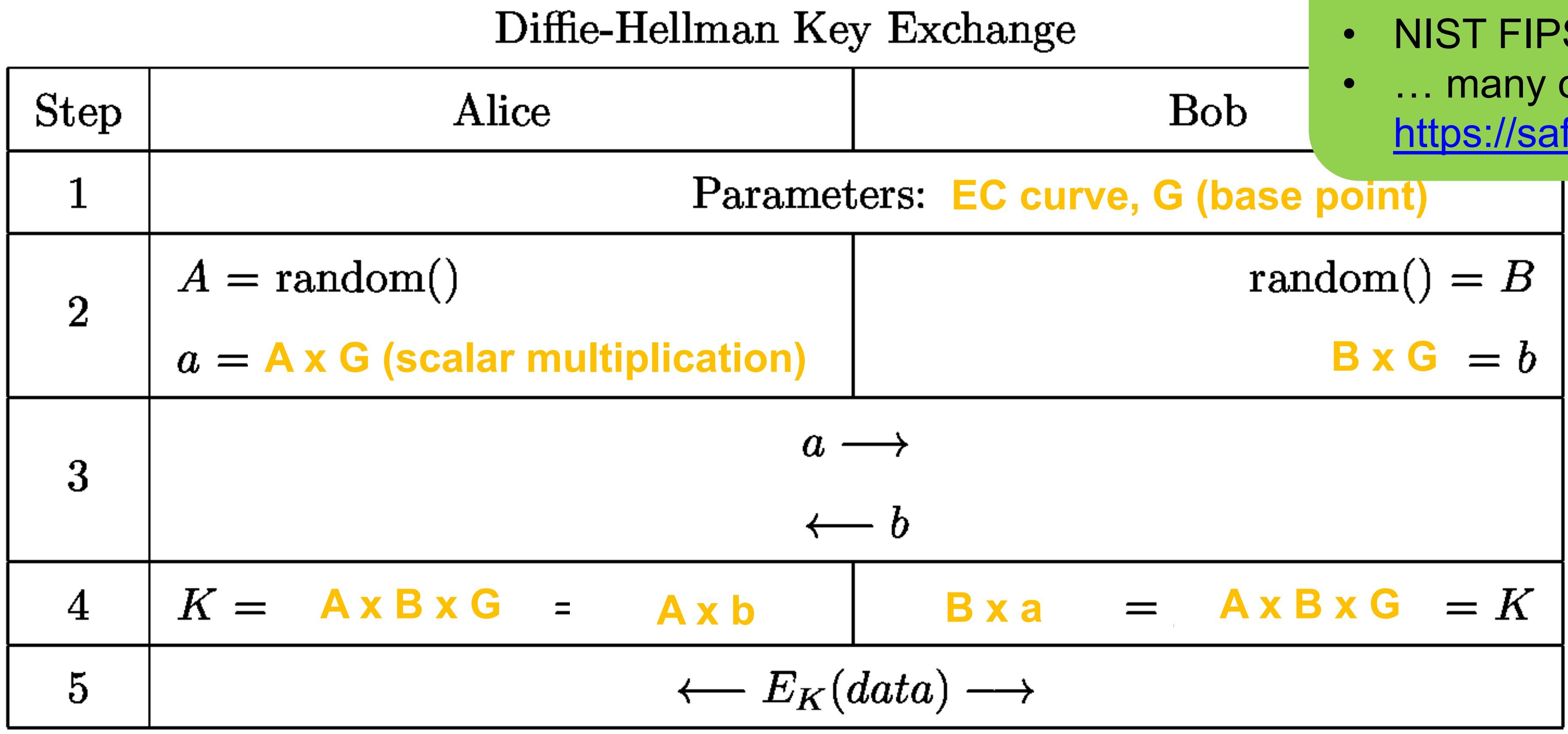
http://www.themccallums.org/nathaniel/2014/10/27/authenticated-key-exchange-with-speke-or-dh-eke/

Diffie-Hellman in practice

- Be aware of particular `p` and `g`
  - If `g` is widely used with length up to 1024b then precomputation is possible
    - "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/



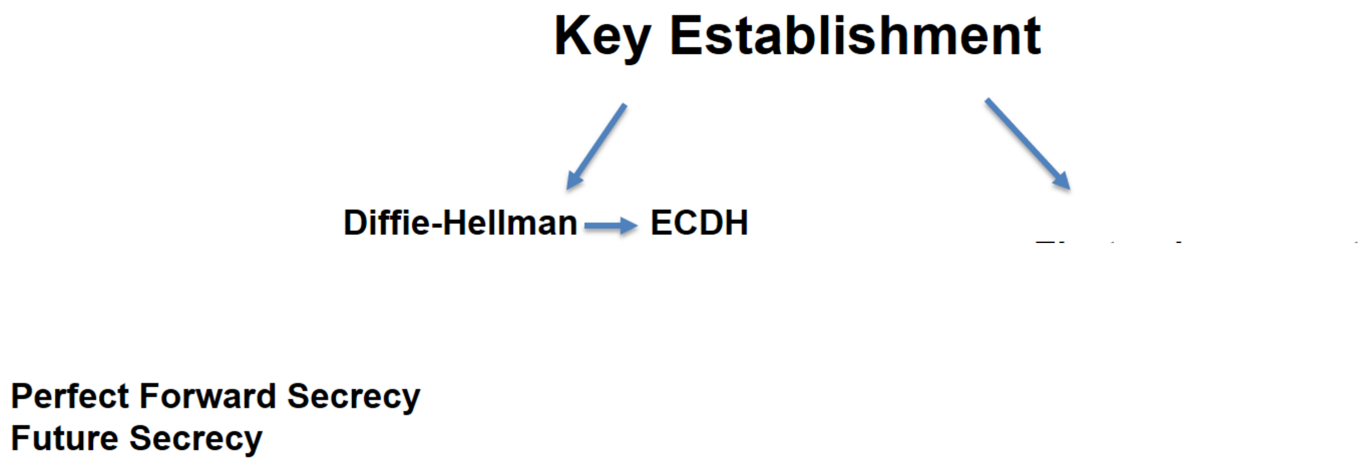
Diffie-Hellman Key Exchange

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
  2. Multiple keys may be required (K_{ENC}, K_{MAC})
- Is vulnerable to man-in-the-middle attack (MitM)
  – Attacker runs separate DH with A and B simultaneously
  – (Unless a and b are authenticated)
- DH can be used as basis for *Password-Authenticated Key Exchange*
- DH can be used as basis for *Forward/Backward/Future secrecy*



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

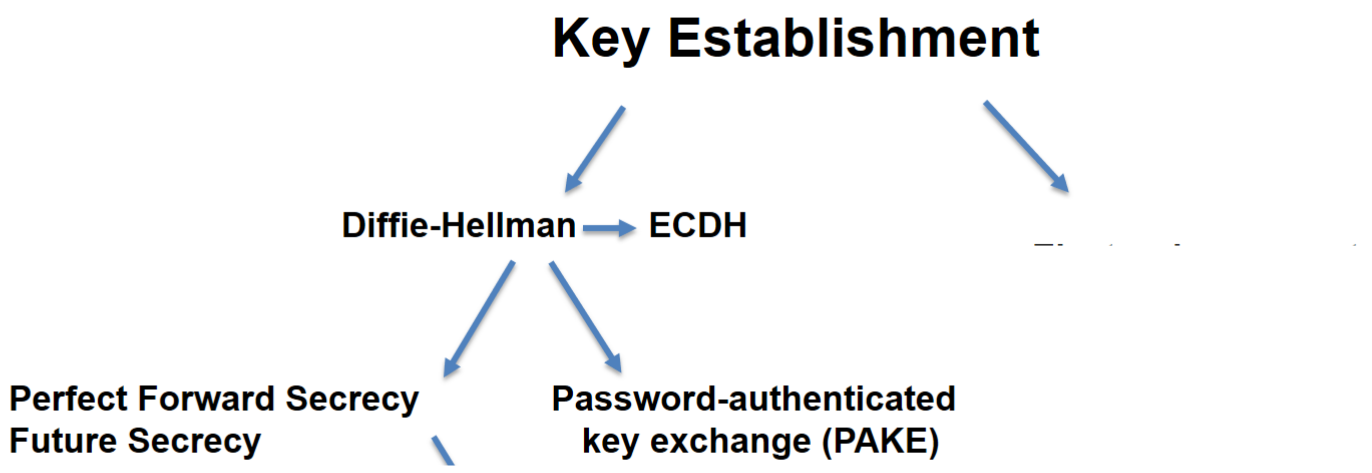
Must not derive future keys deterministically

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
- 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)



PASSWORD-AUTHENTICATED KEY EXCHANGE (PAKE)

PAKE protocols - motivation

- Diffie-Hellman can be used for key establishment
  - Authentication ca be added via pre-shared (longterm) 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]

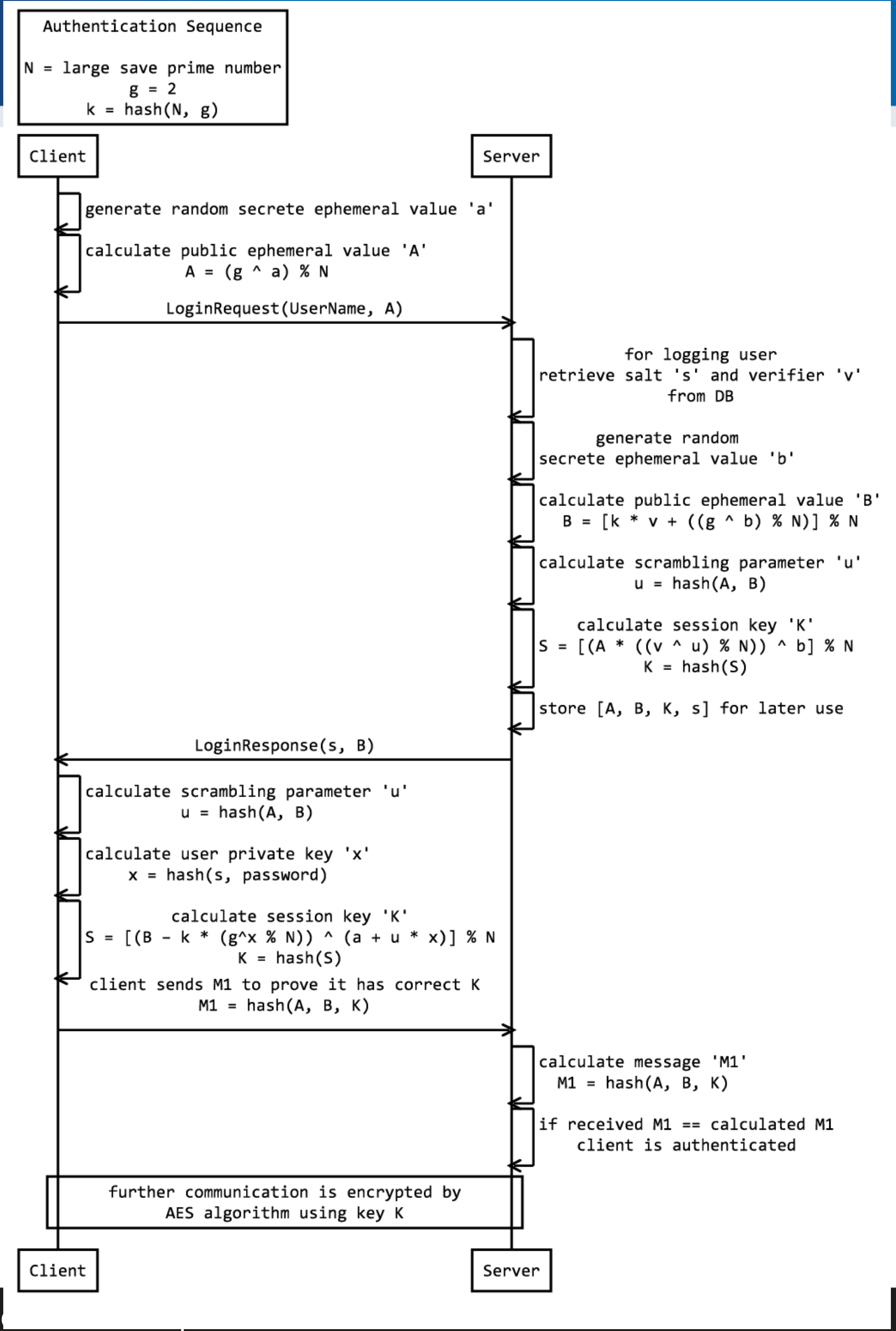
Step	Alice	Bob
1	Shared Secret: $S = H(password)$	
2	Parameters: p, g	
3	$A = \text{random}()$ $a = g^A \pmod p$	$\text{random}() = B$ $g^B \pmod p = b$
4a	$E_S(a) \longrightarrow$ $\longleftarrow E_S(b)$	
4b	$a \longrightarrow$ $\longleftarrow E_S(b)$	
4c	$E_S(a) \longrightarrow$ $\longleftarrow b$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

Secure Remote Password protocol (SRP), [aPAKE]

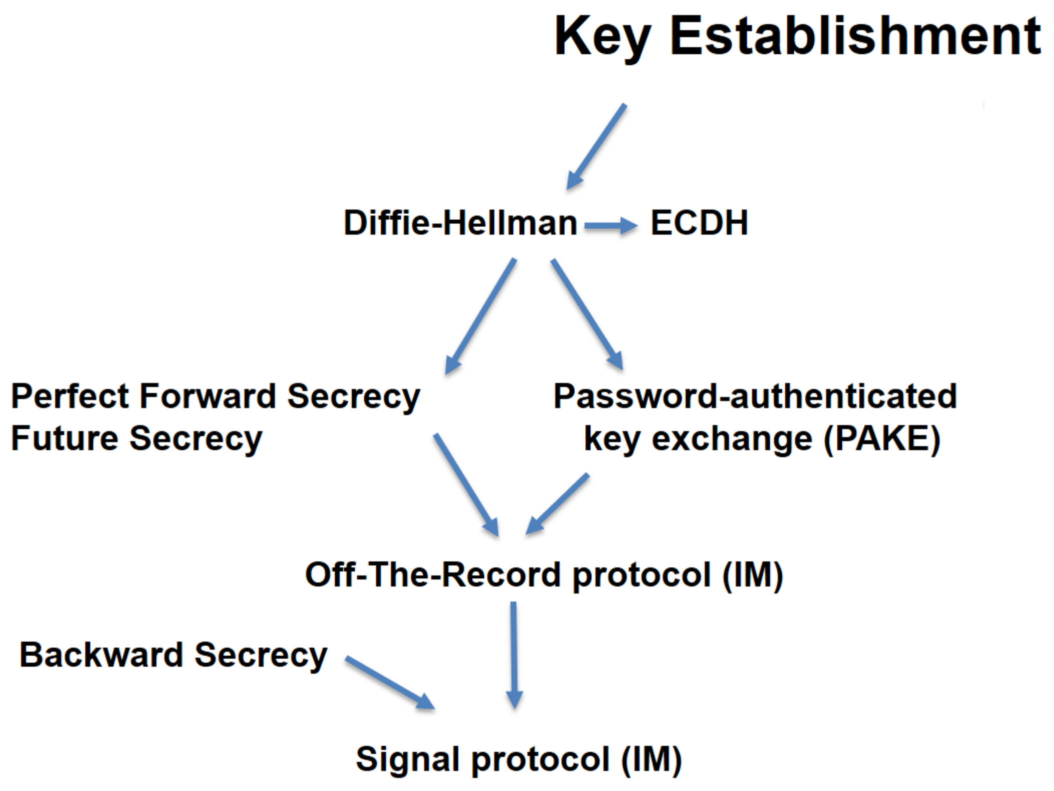
- Earlier Password-Authenticated Key Exchange protocols (PAKEs) were patented: EKE, SPEKE… (patent expired in 2017)
- Secure Remote Password protocol (SRP) 1998
  - Designed to work around existing patents
  - Royalty free, open license (Standford university), basis for multiple RFCs
  - Several revisions 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)



```
Authentication Sequence

N = large save prime number
g = 2
k = hash(N, g)

Client                                              Server

Client: generate random secrete ephemeral value 'a'
Client: calculate public ephemeral value 'A'
        A = (g ^ a) % N
Client -> Server: LoginRequest(UserName, A)
Server: for logging user
        retrieve salt 's' and verifier 'v'
        from DB
Server: generate random
        secrete ephemeral value 'b'
Server: calculate public ephemeral value 'B'
        B = [k * v + ((g ^ b) % N)] % N
Server: calculate scrambling parameter 'u'
        u = hash(A, B)
Server: calculate session key 'K'
        S = [(A * ((v ^ u) % N)) ^ b] % N
        K = hash(S)
Server: store [A, B, K, s] for later use
Server -> Client: LoginResponse(s, B)
Client: calculate scrambling parameter 'u'
        u = hash(A, B)
Client: calculate user private key 'x'
        x = hash(s, password)
Client: calculate session key 'K'
        S = [(B - k * (g^x % N)) ^ (a + u * x)] % N
        K = hash(S)
Client -> Server: client sends M1 to prove it has correct K
        M1 = hash(A, B, K)
Server: calculate message 'M1'
        M1 = hash(A, B, K)
Server: if received M1 == calculated M1
        client is authenticated
Client, Server: further communication is encrypted by
        AES algorithm using key K

Client                                              Server
```

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

SRP is unnecessarily complex to work around existing patens



SECURE INSTANT MESSAGING

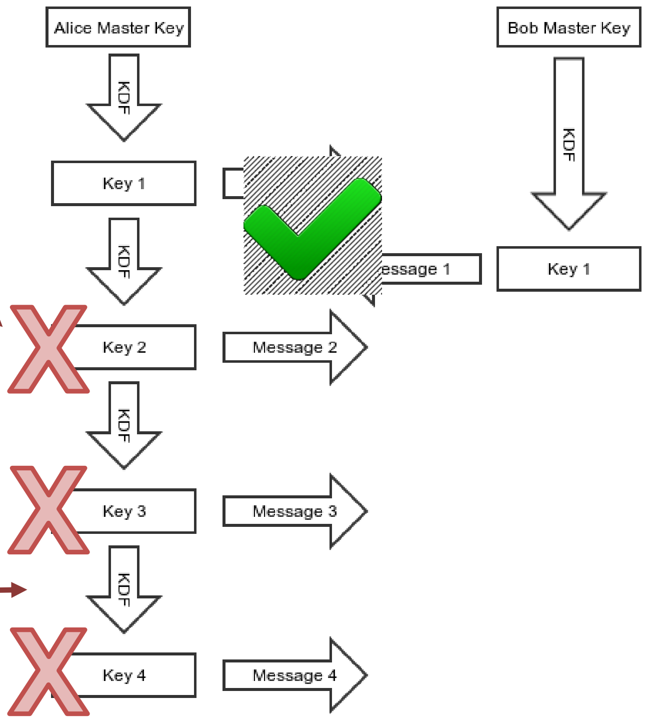
"Toy" protocol for protection of instant messaging

https://signal.org/blog/advanced-ratcheting/

- Relatively short sessions with multiple messages
- Perfect forward secrecy
  - – Ephemeral DH to establish Alice/Bob master keys
    - Past keys/messages are secure
- Derive next key within session by KDF (hash)

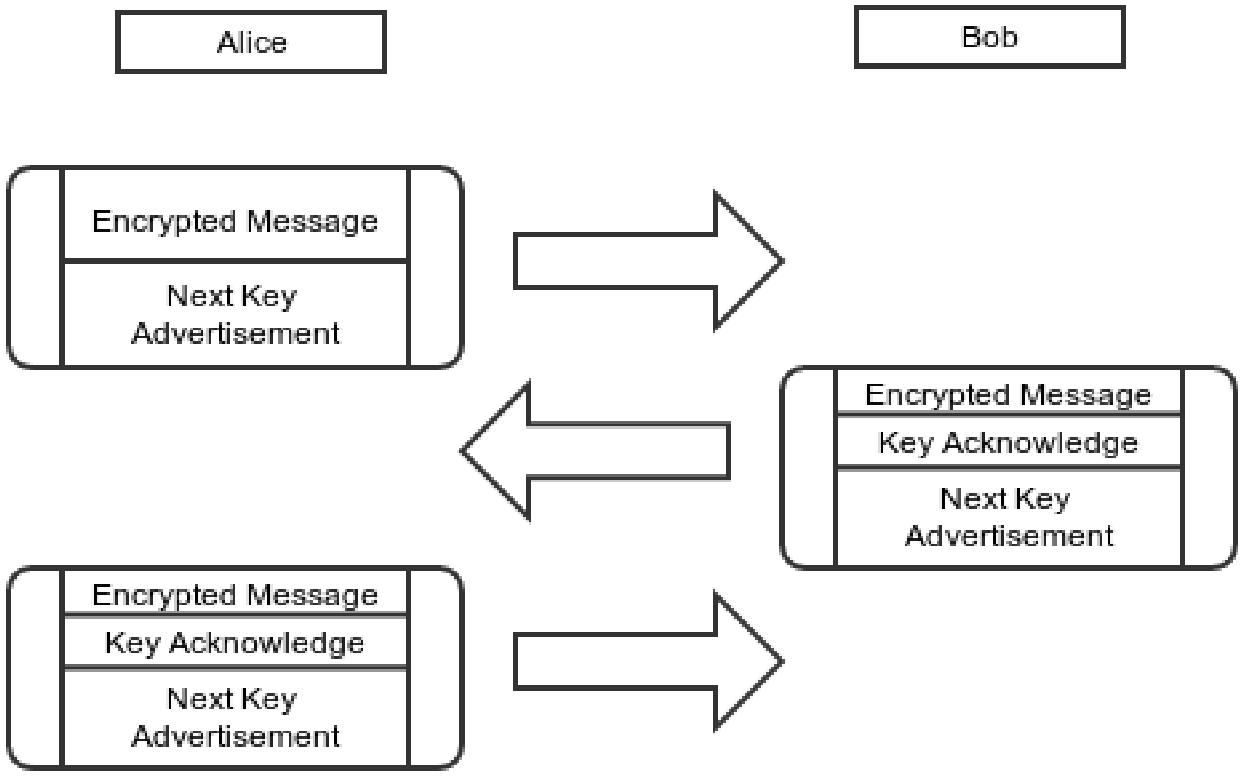
Key 2 is compromised

All subsequent session keys now compromised



- We also need "Future" secrecy
  - – Automatic self-healing after key compromise
  - – Next key must NOT be deterministically generated from previous ones

“Ratcheting” == new DH exchange for every message



https://signal.org/blog/advanced-ratcheting/

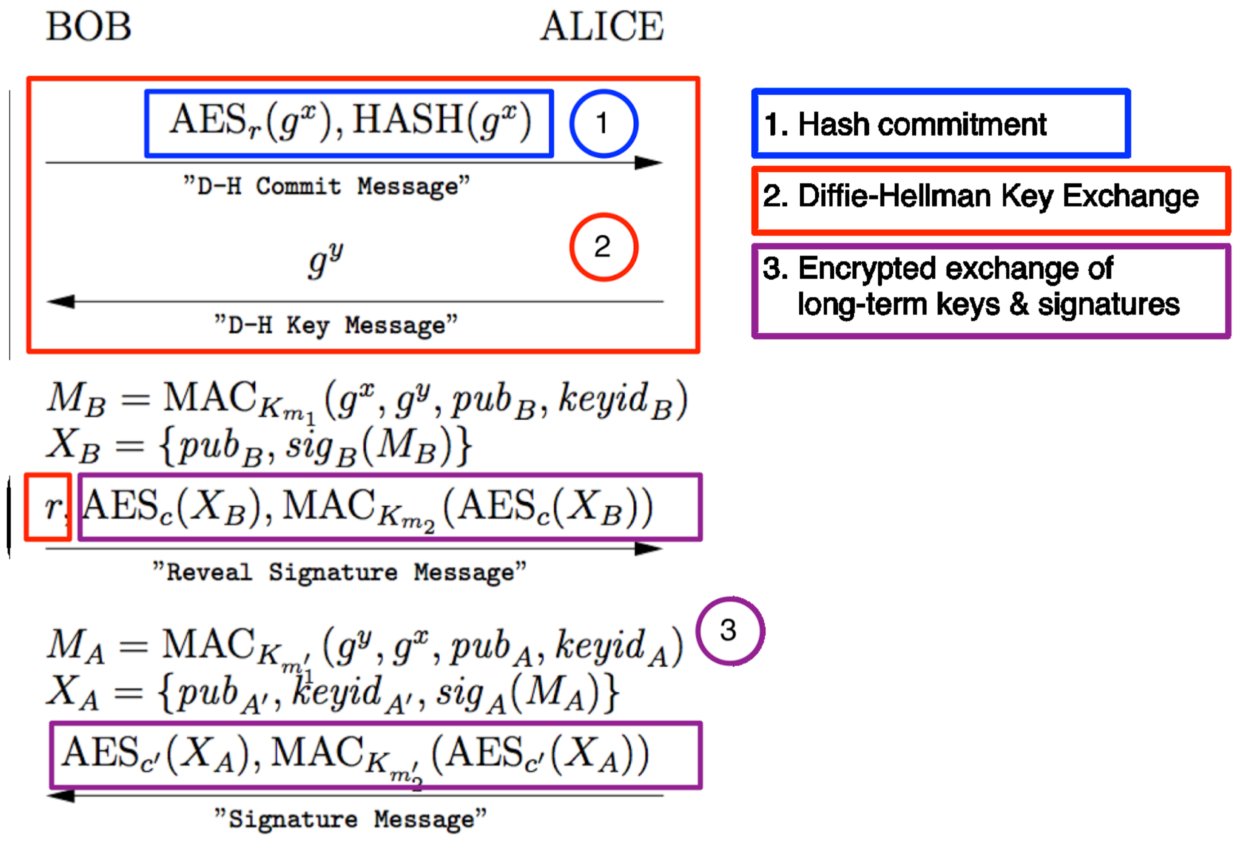
Off-The-Record Messaging (OTR), 2004

- Protocol for protection of instant messaging
  - Establish session, communicate, close (minutes/hours)
- Perfect forward secrecy (using ephemeral DH keys)
  - Also “future” secrecy: automatic self-healing after compromise
- OTR “ratcheting” (new DH key for every session & new message)
- Plausible deniability of messages
  - Message MAC is computed, message send and received
  - MAC key used to compute MAC is then publicly broadcast
  - As MAC key is now public, everyone can forge past messages (will not affect legitimate users but can dispute claims of cryptographic message log in court)

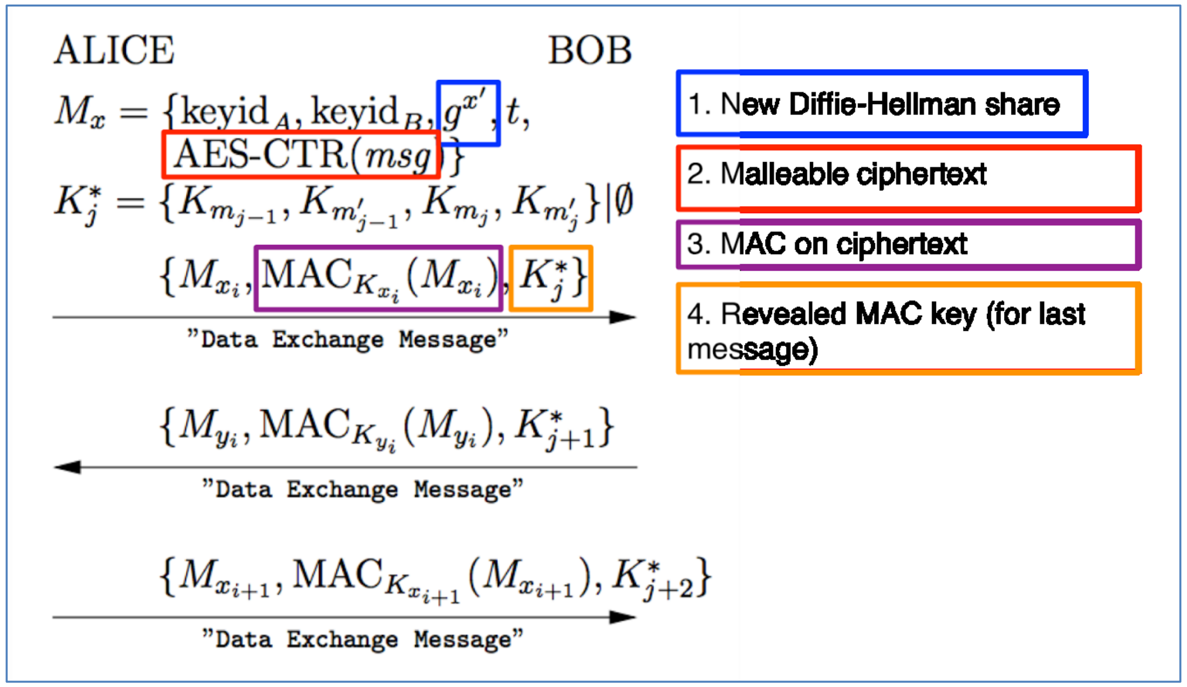
OTR protocol

See https://blog.cryptographyengineering.com/2014/07/26/noodling-about-im-protocols/ for details

Key exchange


BOB
ALICE
$\mathrm{AES}_r(g^x), \mathrm{HASH}(g^x)$
1
"D-H Commit Message"
g^y
2
"D-H Key Message"
$M_B = \mathrm{MAC}_{K_{m_1}}(g^x, g^y, pub_B, keyid_B)$
$X_B = \{pub_B, sig_B(M_B)\}$
$r, \mathrm{AES}_c(X_B), \mathrm{MAC}_{K_{m_2}}(\mathrm{AES}_c(X_B))$
"Reveal Signature Message"
$M_A = \mathrm{MAC}_{K_{m_1'}}(g^y, g^x, pub_A, keyid_A)$
3
$X_A = \{pub_{A'}, keyid_{A'}, sig_A(M_A)\}$
$\mathrm{AES}_{c'}(X_A), \mathrm{MAC}_{K_{m_2'}}(\mathrm{AES}_{c'}(X_A))$
"Signature Message"
1. Hash commitment
2. Diffie-Hellman Key Exchange
3. Encrypted exchange of long-term keys & signatures

Message exchange


ALICE
BOB
$M_x = \{\mathrm{keyid}_A, \mathrm{keyid}_B, g^{x'}, t, \mathrm{AES\text{-}CTR}(msg)\}$
$K_j^* = \{K_{m_{j-1}}, K_{m'_{j-1}}, K_{m_j}, K_{m'_j}\} | \emptyset$
$\{M_{x_i}, \mathrm{MAC}_{K_{x_i}}(M_{x_i}), K_j^*\}$
"Data Exchange Message"
$\{M_{y_i}, \mathrm{MAC}_{K_{y_i}}(M_{y_i}), K_{j+1}^*\}$
"Data Exchange Message"
$\{M_{x_{i+1}}, \mathrm{MAC}_{K_{x_{i+1}}}(M_{x_{i+1}}), K_{j+2}^*\}$
"Data Exchange Message"
1. New Diffie-Hellman share
2. Malleable ciphertext
3. MAC on ciphertext
4. Revealed MAC key (for last message)

OTR – some problems

- How to work with asynchronous messages?
  - OTR designed for instant messaging with short sessions
- What if out-of-order message is received?
  - OTR has counter to prevent replay
- Window of compromise is extended
  - Decryption key cannot be deleted until message arrives
- …
- Systematization of Knowledge: Secure Messaging (2015)
  - Systematic mapping of Secure Messaging protocols
  - http://www.ieee-security.org/TC/SP2015/papers-archived/6949a232.pdf

SIGNAL PROTOCOL

The Signal protocol

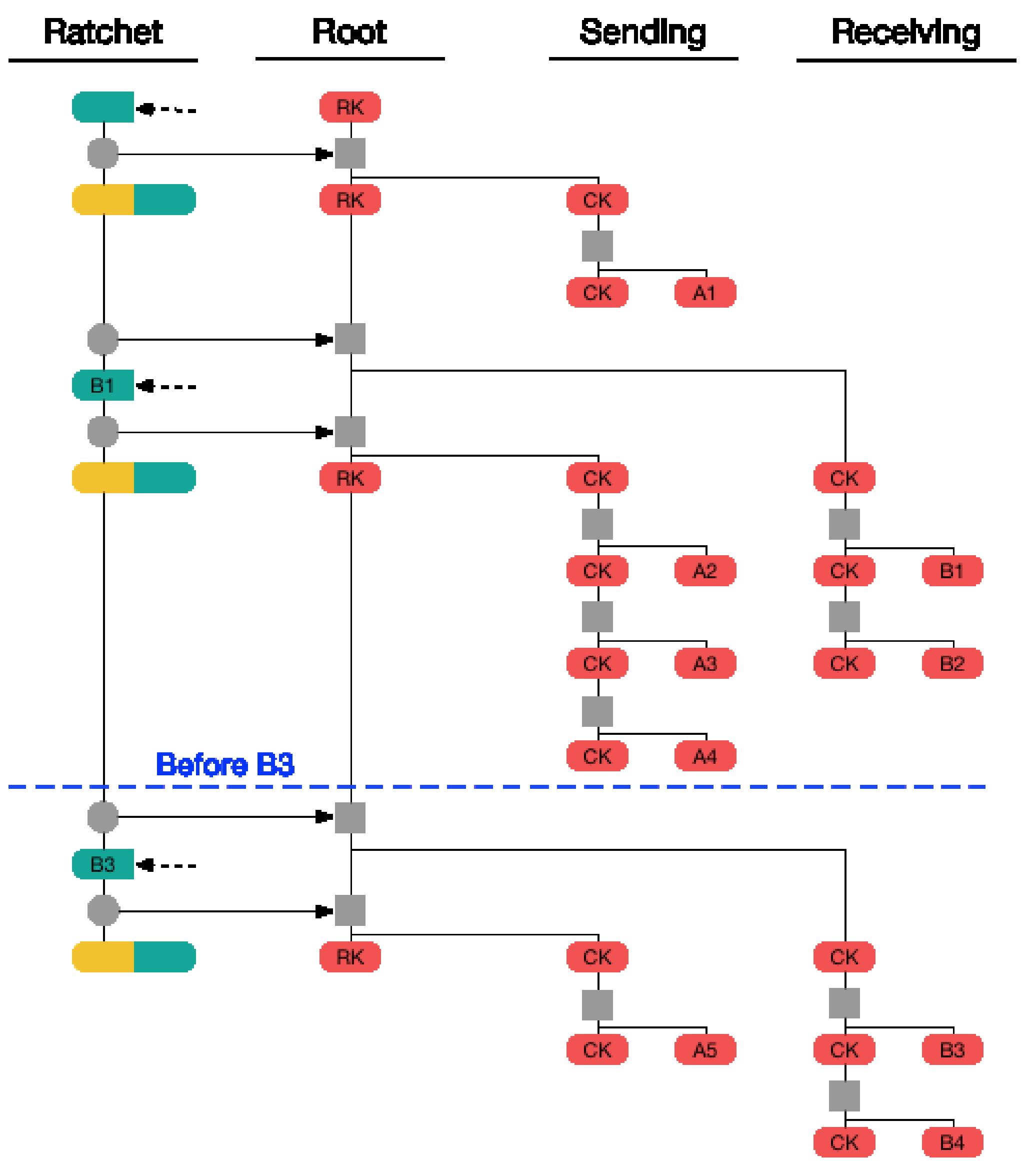
- State-of-the-art of instant messaging protocols
  - Used in Signal, WhatsApp, Facebook Messenger, Google Allo…
- The Signal protocol provides:
  - confidentiality, integrity, message authentication,
  - participant consistency, destination validation,
  - forward secrecy, backward secrecy (aka future secrecy)
  - causality preservation, message unlinkability, message repudiation, participation repudiation and asynchronicity
  - end-to-end encrypted group chats
- Requires servers (but servers are untrusted wrt message privacy/integrity)
  - relaying of messages and storage of public key material
- ECDH with Curve25519, AES-256, HMAC-SHA256

The Signal protocol implementation

- Authentication of users: 1) Trust on first use 2) Trusted party (PKI) 3) Fingerprint check using other channel (hex, QR code…)
- Protection of messages
  - Perfect forward secrecy and backward secrecy (ratcheting)
  - New DH for (almost) every message (announced in the previous one)
  - Message key derived both from long-term key and chain key
  - Authenticated Encryption with deniability (MAC key broadcasted later)
- Protection of metadata (but no strong anonymity such as in Tor)
  - Message delivery time and communicating parties available
  - Service provider may choose to keep or delete this information
- Private contact discovery using Intel SGX
  - https://signal.org/blog/private-contact-discovery/

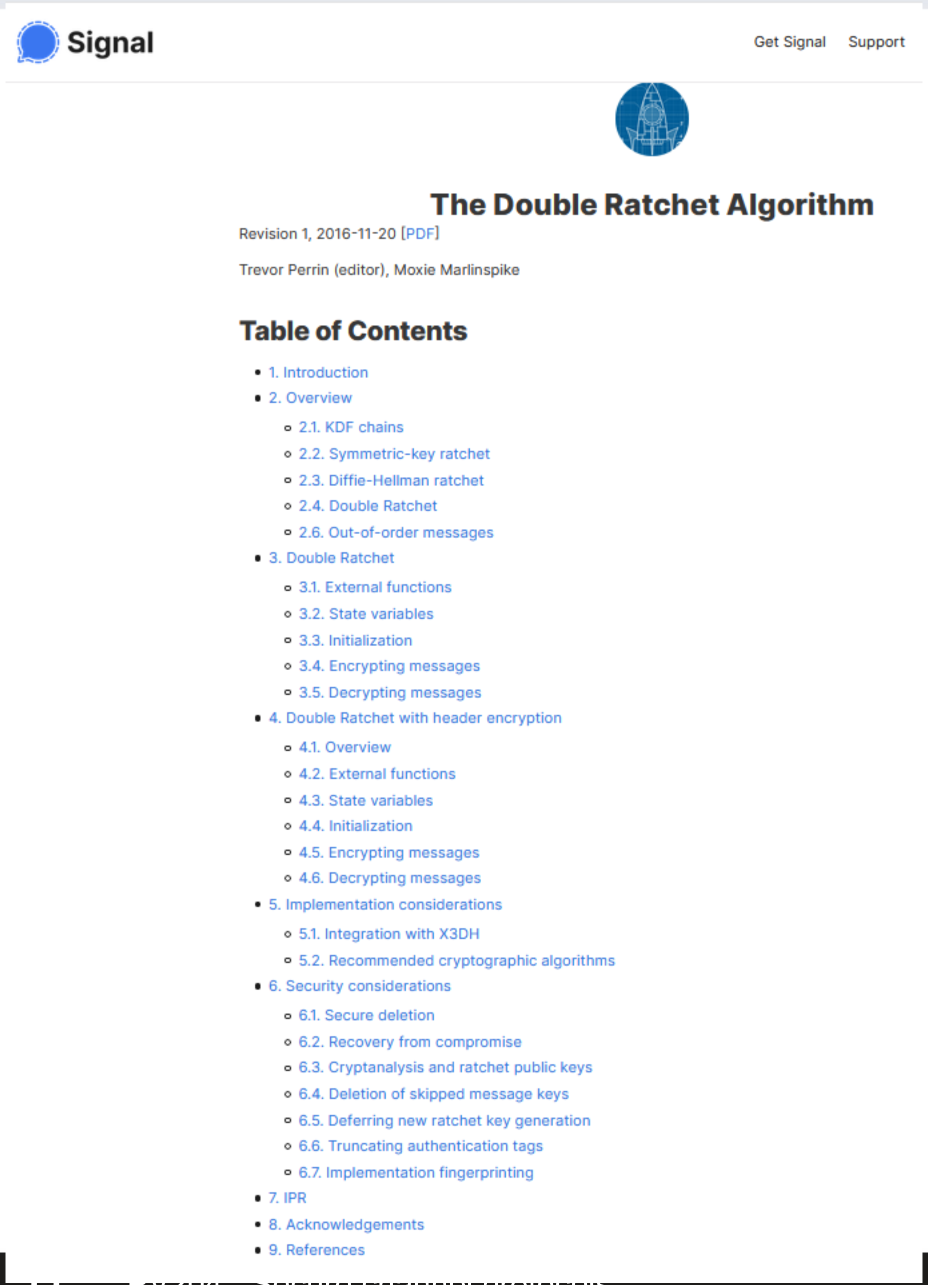
Message keys in Signal

- Basic trick: combine frequent ECDH and has
- Root key(s) (RK)
  - Established from last ECDH ratchet and previous R
- Chain key(s) (CK)
  - Established from the most recent RK + hash chain
  - KDF to derive next CK = HMAC-HASH(CK, "1")
- Message key(s) (MK)
  - Derived from CK as MK = HMAC-HASH(CKs, "0")
  - Message A_x encrypted by MK_x
- RK&CK compromise is “healed” by next ECD
- Out-of-order messages by storage of corresp



CROCS

https://signal.org/docs/specifications/doubleratchet/



Signal Get Signal Support

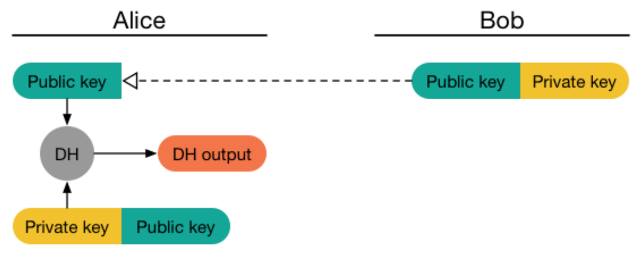
The Double Ratchet Algorithm

Revision 1, 2016-11-20 [PDF]

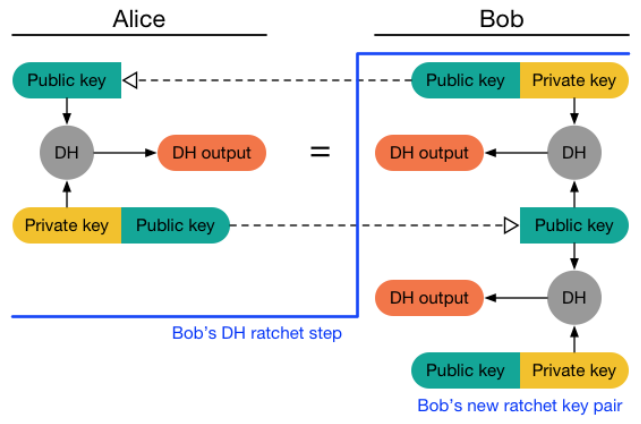
Trevor Perrin (editor), Moxie Marlinspike

Table of Contents

Signal Get Signal Support Blog Developers Caree



Alice's initial messages advertise her ratchet public key. Once Bob receives one of these messages, Bob performs a DH ratchet step: He calculates the DH output between Alice's ratchet public key and his ratchet private key, which equals Alice's initial DH output. Bob then replaces his ratchet key pair and calculates a new DH output:



NOISE PROTOCOL FRAMEWORK

Noise protocol framework (https://noiseprotocol.org/)

- Do you have exactly same situation as Signal?
  - Then use Signal’s code, but: different usage scenario, unnecessarily large code base…
- What if different usage scenario is required with some changes needed?
  - Custom changes to protocol are always risky
  - (Signal is formally verified, but change may break the proof + change in assumptions)
- And formally verified protocols is not enough – we need also secure implementation!

- Ideally: custom, but formally verified protocol + generator of its code

Noise protocol framework (https://noiseprotocol.org/)

- Ideally, formally verified protocol + generator of code
- Noise protocol framework is exactly that
  - T. Perrin (co-author of Signal's double-ratchet protocol)
  - https://noiseprotocol.org/noise.pdf + generator (https://noiseexplorer.com/)
  - Used by WhatsApp since October 2020, Bitcoin Lighting protocol, I2P anonymity network, WireGuard VPN

1. Protocol designer to describe what needs using simple language or picks from already existing patterns (many of them)
2. Execute formal verification (ProVerif) to verify protocol claims
   - E.g., claim: same shared key, one party authenticated for passive attacker
3. Run code generator to get complete code in target language

CROCS

Some existing Noise patterns

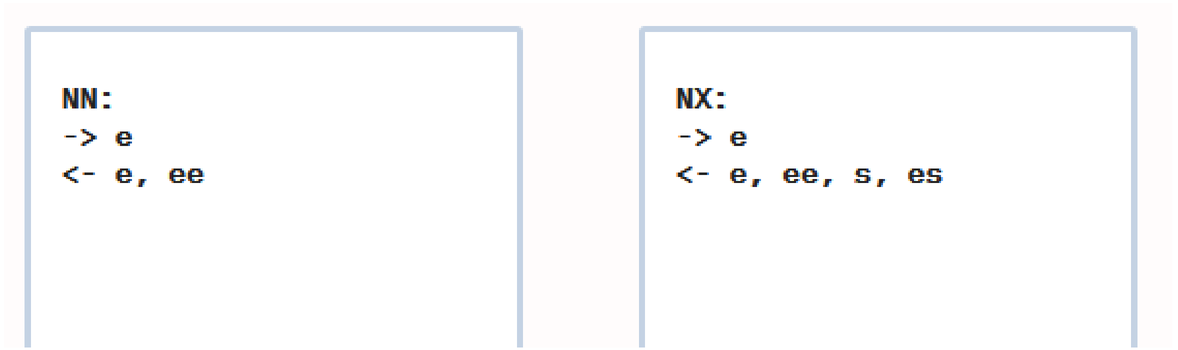
The first character refers to the initiator's static key:

- N = No static key for initiator
- K = Static key for initiator Known to responder
- X = Static key for initiator Xmitted ("transmitted") to responder
- I = Static key for initiator Immediately transmitted to responder, despite reduced or absent identity hiding

The second character refers to the responder's static key:

- N = No static key for responder
- K = Static key for responder Known to initiator
- X = Static key for responder Xmitted ("transmitted") to initiator

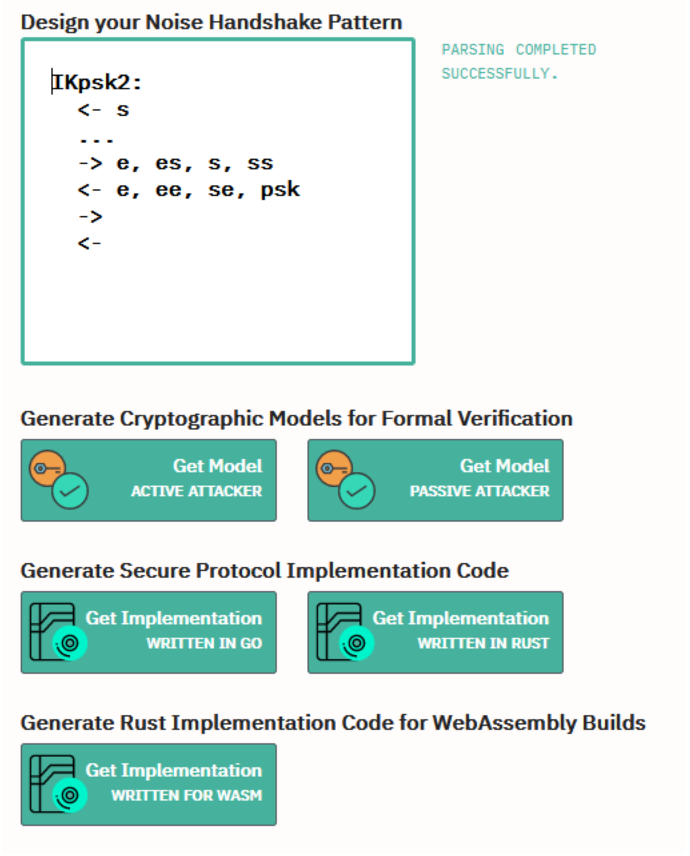
https://noiseexplorer.com/patterns/NN/
https://noiseexplorer.com/patterns/NX/



```
NN:
-> e
<- e, ee
```

```
NX:
-> e
<- e, ee, s, es
```

https://noiseexplorer.com/patterns/IKpsk2/



Design your Noise Handshake Pattern

```
IKpsk2:
  <- s
  ...
  -> e, es, s, ss
  <- e, ee, se, psk
  ->
  <-
```

PARSING COMPLETED SUCCESSFULLY.

Generate Cryptographic Models for Formal Verification

Get Model ACTIVE ATTACKER
Get Model PASSIVE ATTACKER

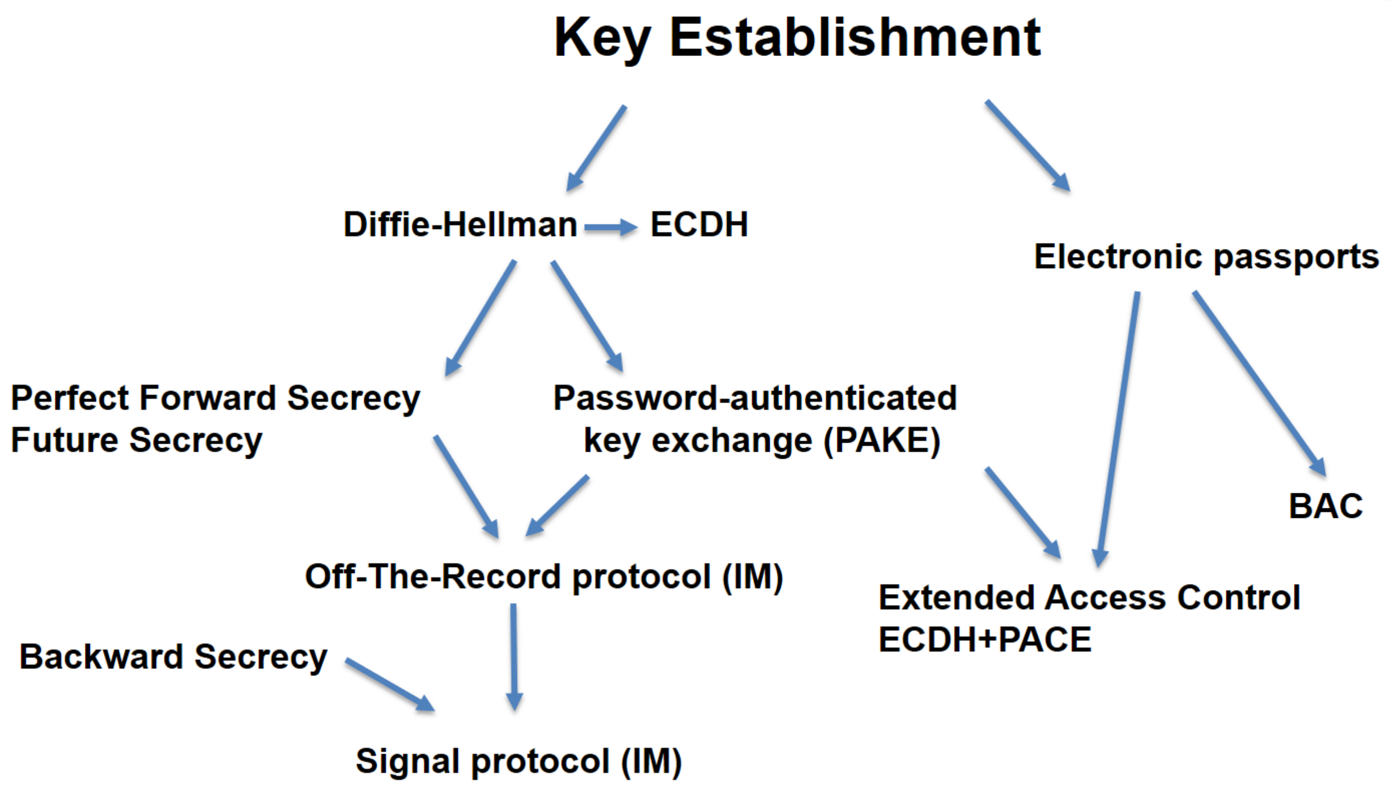
Generate Secure Protocol Implementation Code

Get Implementation WRITTEN IN GO
Get Implementation WRITTEN IN RUST

Generate Rust Implementation Code for WebAssembly Builds

Get Implementation WRITTEN FOR WASM

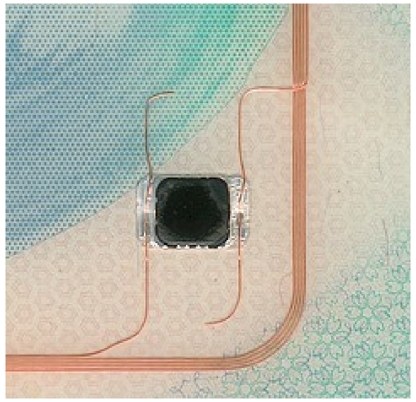
- Noise protocols use set of rules:
  - If you already have some key, use it immediately to encrypt all subsequent messages
  - When new entropy is available (ECDH), update current state (keys) => forward&backward secrecy
  - Use AEAD for all message encryption
  - …



ELECTRONIC PASSPORTS AND CITIZEN ID CARDS

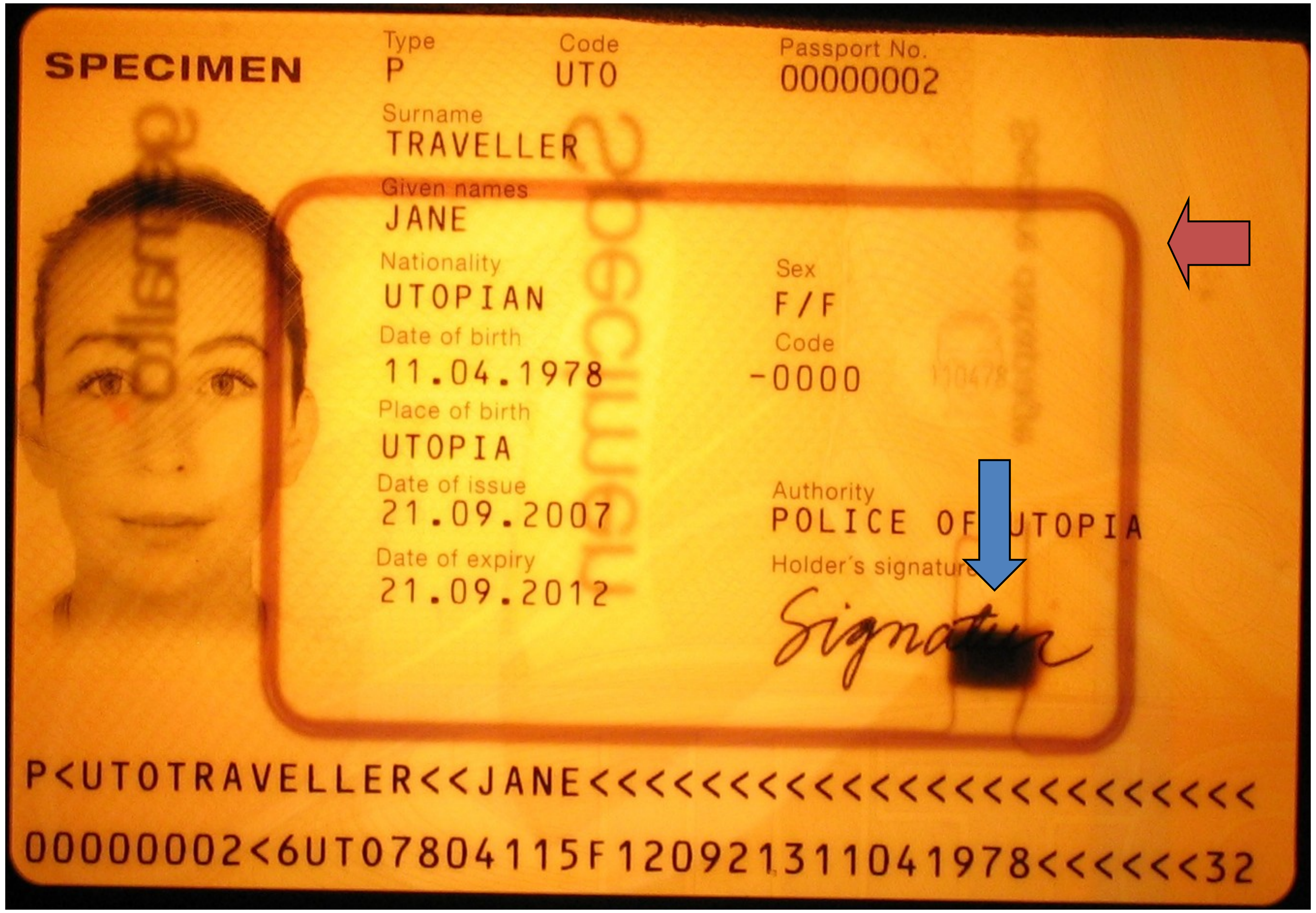
Credit: Slides partially based on presentation by Zdenek Říha

Passports of the first generation



- Electronic passport
  - Classical passport booklet + passive contactless smartcard (ISO14443, communication distance 0-10 cm)
  - Chip & antenna integrated in a page or cover
- Technical specification standardized by ICAO
  - Standard 9303, 6th edition
  - References many ISO standards
- Data is organised in 16 data groups (DG) and 2 meta files
  - DG1-DG16, EF.COM, EF.SOD
  - Mandatory is DG1 (MRZ), DG2 (photo), EF.COM and EF.SOD (passive authentication)

Chip and antenna



SPECIMEN

Type
P

Code
UTO

Passport No.
00000002

Surname
TRAVELLER

Given names
JANE

Nationality
UTOPIAN

Sex
F/F

Date of birth
11.04.1978

Code
-0000

Place of birth
UTOPIA

Date of issue
21.09.2007

Authority
POLICE OF UTOPIA

Date of expiry
21.09.2012

Holder's signature
[illegible]

P<UTOTRAVELLER<<JANE<<<<<<<<<<<<<<<<<<<<<<<<<

00000002<6UTO7804115F1209213110419 78<<<<<<32

Data groups

Data group	Stored data
DG1	**Machine readable zone (MRZ)**
DG2	**Biometric data: face**
DG3	Biometric data: fingerprints
DG4	Biometric data: iris
DG5	Picture of the holder as printed in the passport
DG6	Reserved for future use
DG7	Signature of the holder as printed in the passport
DG8	Encoded security features – data features
DG9	Encoded security features – structure features
DG10	Encoded security features – substance features
DG11	Additional personal details (address, phone)
DG12	Additional document details (issue date, issued by)
DG13	Optional data (anything)
DG14	Data for securing secondary biometrics (EAC)
DG15	Active Authentication public key info
DG16	Next of kin

Protocols used in ePassports I.

I. Authentication of inspection system to chip [BAC]
   - Read basic digital data from chip (MRZ, photo)
   - SG: Passport provides basic data only to local terminal with physical access to passport
   - S: Auth. SCP, sym. crypto keys derived from MRZ [BAC]

II. Authorized access to more sensitive chip data
   - SG: Put more sensitive data on chip (fingerprint, iris), but limit availability only to inspection systems of trustworthy countries
   - S: Challenge-response auth. protocol [EAC,EAC-PACE], PKI + cross-signing between trustworthy states [EAC]

Protocols used in ePassports II.

III. Genuine data on passport
   - SG: Are data on passport unmodified?
   - S: digital signatures, PKI [passive authentication]

IV. Authentication of chip to inspection system
   - SG: Is physical chip inside passport genuine?
   - S: Challenge-response authentication protocol [AA, EAC-PACE]

V. Transfer data between chip and IS securely
   - SG: attacker can’t eavesdrop/modify/replay
   - S: secure channel [EAC, EAC-PACE]

How Signal and ePassports compare?

- Completely different usage scenario
  - Instant messaging vs. person/terminal authentication
  - Frequent software updates possible vs. 15 years passport validity
- Different trust relations and participants structure
  - N friends vs. many partially or fully distrusting participants
  - Mostly online vs. mixed offline/online (even without clock!)
- Underlying cryptographic primitives are shared
  - Forward secrecy, ECDH, AES, SHA-2…
  - Ratcheting and deniability not necessary for ePass

STILL WANT TO DESIGN OWN PROTOCOL? ☺

Design of cryptographic protocols

- Don’t design own cryptographic protocols
  - Use existing and well-studied protocols (TLS, EAC-PACE…)
  - Don’t remove “unnecessary” parts of existing protocols
- Don’t implement existing/your protocol (if possible)
  - Potential for error, implementation attacks…, use existing implementations
- Follow all required checks on incoming messages
  - Verification of cryptograms, check for revocation…
- But more likely you will need to design own protocol than to design own crypto algorithm
  - Always use existing protocol if possible

Conclusions

- Design of (secure) protocols is very hard
  - Understand what are your requirements
  - Use existing protocols, e.g., TLS, Signal or EAC-PACE
  - Use existing implementations (very hard to implement securely)
- Resiliency against compromise of long-term secrets is crucial (forward secrecy)
- Strong session keys authenticated by weak passwords (PAKEs)
- Signal protocol is state-of-the-art and widely deployed (Instant messaging)
- Electronic passport uses variety of protocols (Interesting and complex scenarios)
- Mandatory reading
  - M. Green, Noodling about IM protocols, http://blog.cryptographyengineering.com/2014/07/noodling-about-im-protocols.html
  - M. Marlinspike, Advanced cryptographic ratcheting https://whispersystems.org/blog/advanced-ratcheting/