

# PV204 Security technologies



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

Petr Švenda  [svenda@fi.muni.cz](mailto:svenda@fi.muni.cz)  [@rngsec](https://twitter.com/rngsec)

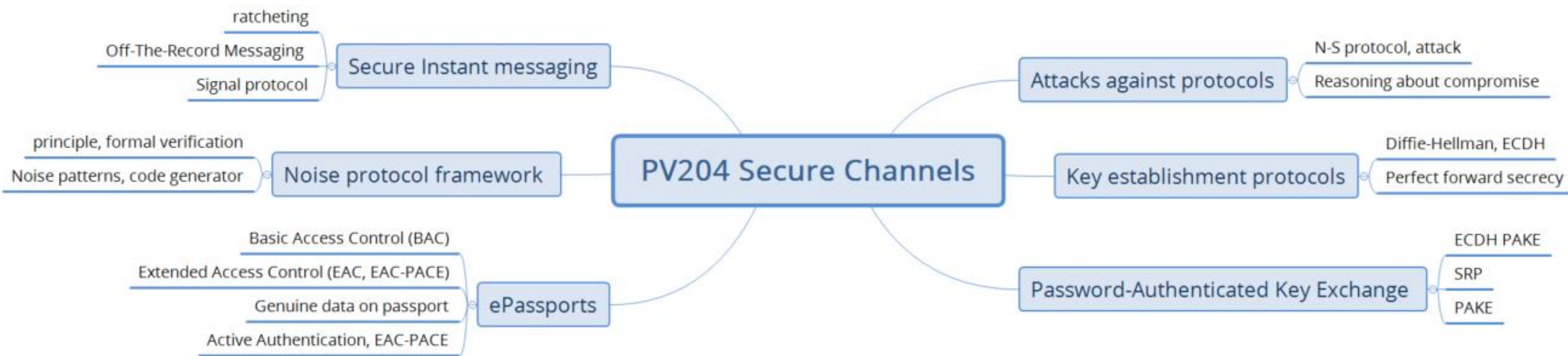
Centre for Research on Cryptography and Security, Masaryk University

*Please provide any corrections and comments here (thank you!):*

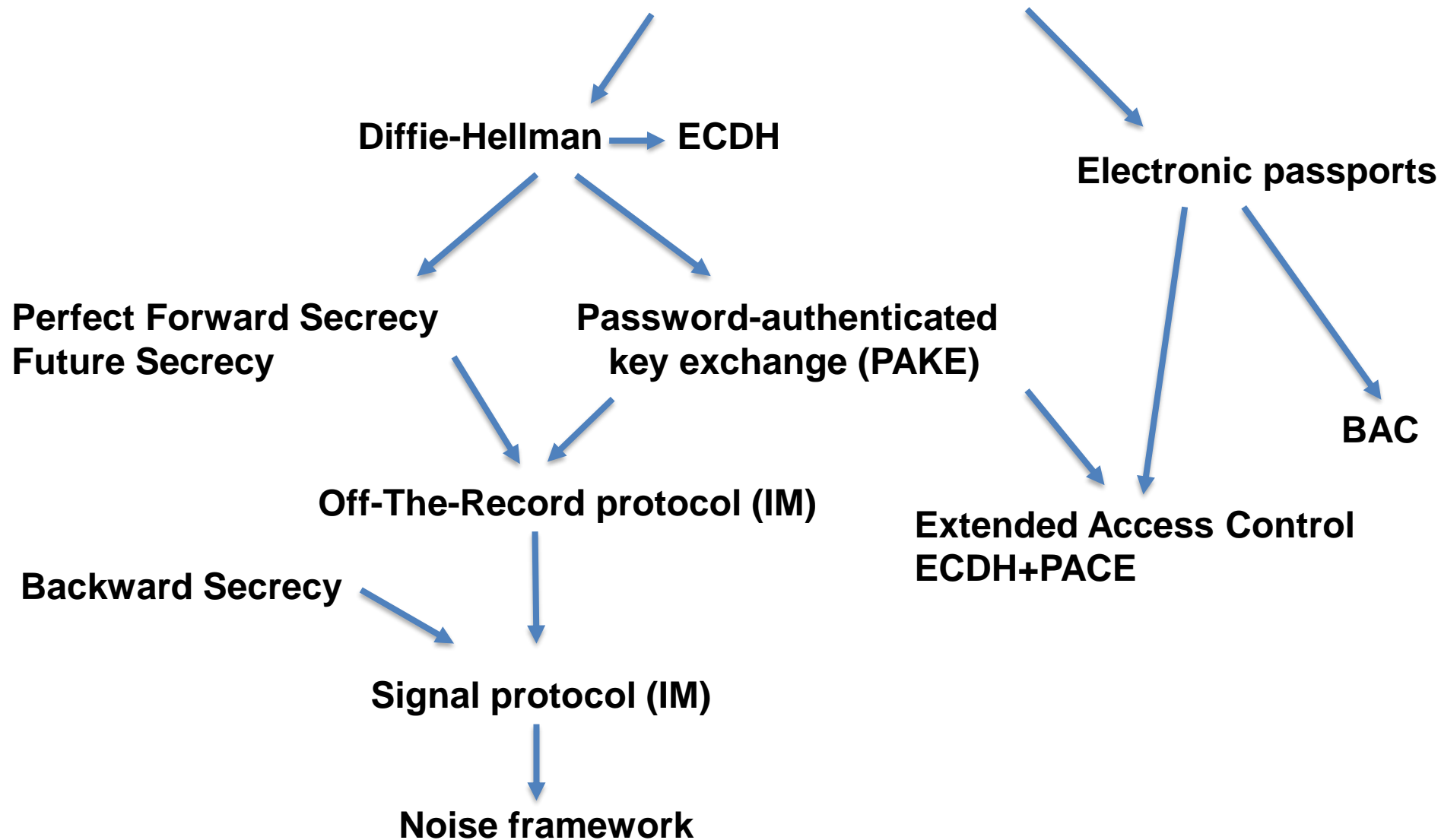
<https://drive.google.com/file/d/1SXnzLqYlucs-96uTx5VeXB2scl5o3lcC/view?usp=sharing>

CRCS

Centre for Research on  
Cryptography and Security



# 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)

## User vs. key-oriented goals

- User-oriented authentication goals
  - Entity authentication
  - Strong entity authentication – fresh entity authentication
  - Mutual authentication
- Key-oriented goals
  - Key establishment
  - Key derivation
  - Implicit key establishment
  - Key confirmation
  - Mutual belief in the key

# Hierarchy of AUTH&KE goals



*Protocols for Authentication and Key Establishment* By Colin Boyd, Anish Mathuria



# Entity authentication

- Entity = user, machine/device
- Something entity knows (password, key...)
- Something entity is (biometrics...)
- Something entity have (smartcard...)
- Multi-factor authentication
  - More than one factor (password + smartcard)
  - Aim to increase attacker's cost to compromise multiple security layers (factors)

# Protocol message components

## 1. Keys

- Long-term keys
- Session keys

## 2. Identifiers

- Protocol principals (Alice, Bob...)

## 3. Nonce(s)

- Random values, timestamps, counters
- Components are combined by crypto mechanism(s)

# 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.  $A \rightarrow B: \{K_{AB}, A\}K_{BS}$
  4.  $B \rightarrow A: \{N_B, A\}K_{AB}$
  5.  $A \rightarrow B: \{N_B - 1\}K_{AB}$

Which part ensures:  
 Authentication  
 Key confirmation  
 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  $N'_B$  generated by B (two additional steps before step 1.)
  - Bob can now check freshness of  $\{K_{AB}, A, N'_B\}K_{BS}$



What 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?



## How to reason about attack and countermeasures?

1. Where does an attack come from (principle)?
  - Understand the principles
2. Different hypothesis for the attack to be practical
  - More ways how to exploit the same weakness
3. Attack's countermeasures by cancel of hypothesis
  - For every way you are aware of
4. Costs and benefits of the countermeasures
  - Cost of the assets protected
  - Cost for an attacker to perform attack
  - Cost of a countermeasure



Important: Consider Break Once, Run Everywhere (BORE)

# What if key is compromised?

- Prevention, detection (is hard), reaction
- Prevention of compromise
  - Limit usage of a key
    - master key → 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)

## Needham–Schroeder protocol: asymmetric

- Simple asymmetric AUTH & KE protocol
- Designed by R. Needham and M. Schroeder (1978)
  - (below is simplified version without PKI server S)

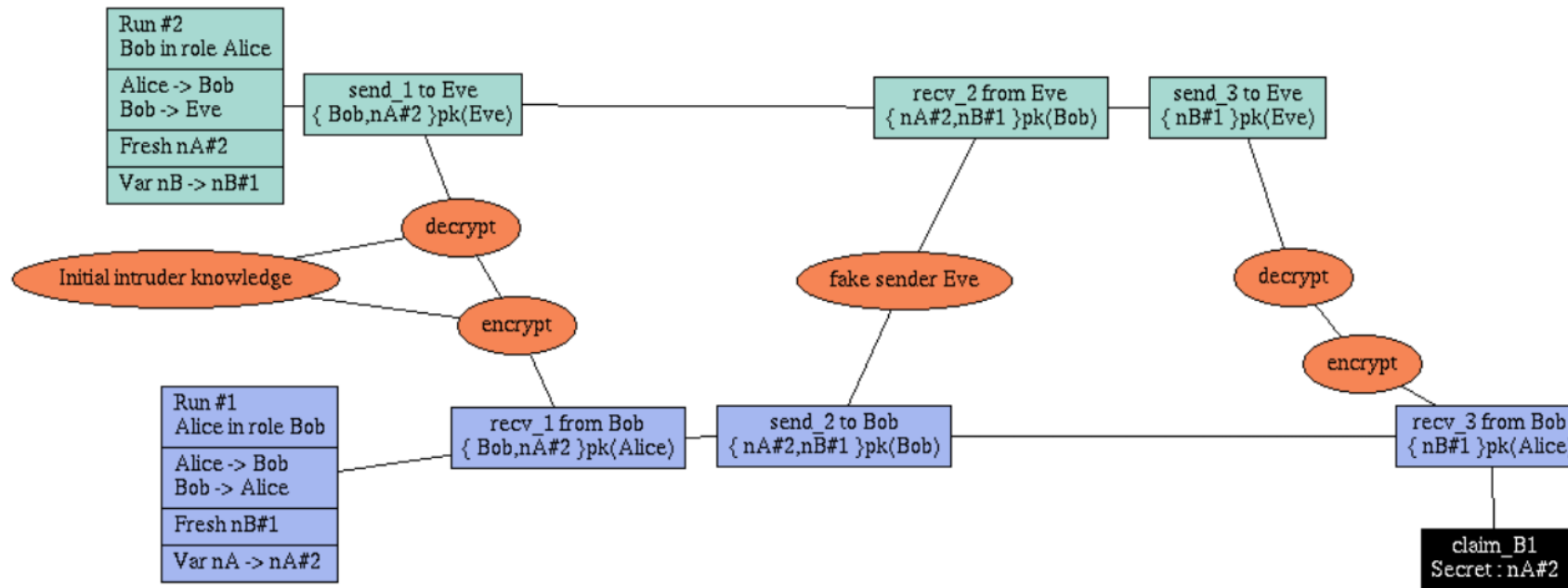
1.  $A \rightarrow B: \{A, N_A\}PK_B$
2.  $B \rightarrow A: \{N_A, N_B\}PK_A$
3.  $A \rightarrow B: \{N_B\}PK_B$

Which part ensures:  
Authentication ?  
Freshness  
Key establishment



Can you spot the problem?

# N-S asymmetric: Problem?



- Discovered by G. Lowe 17 years after using formal verification method/tool

# 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



Is 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**

## References

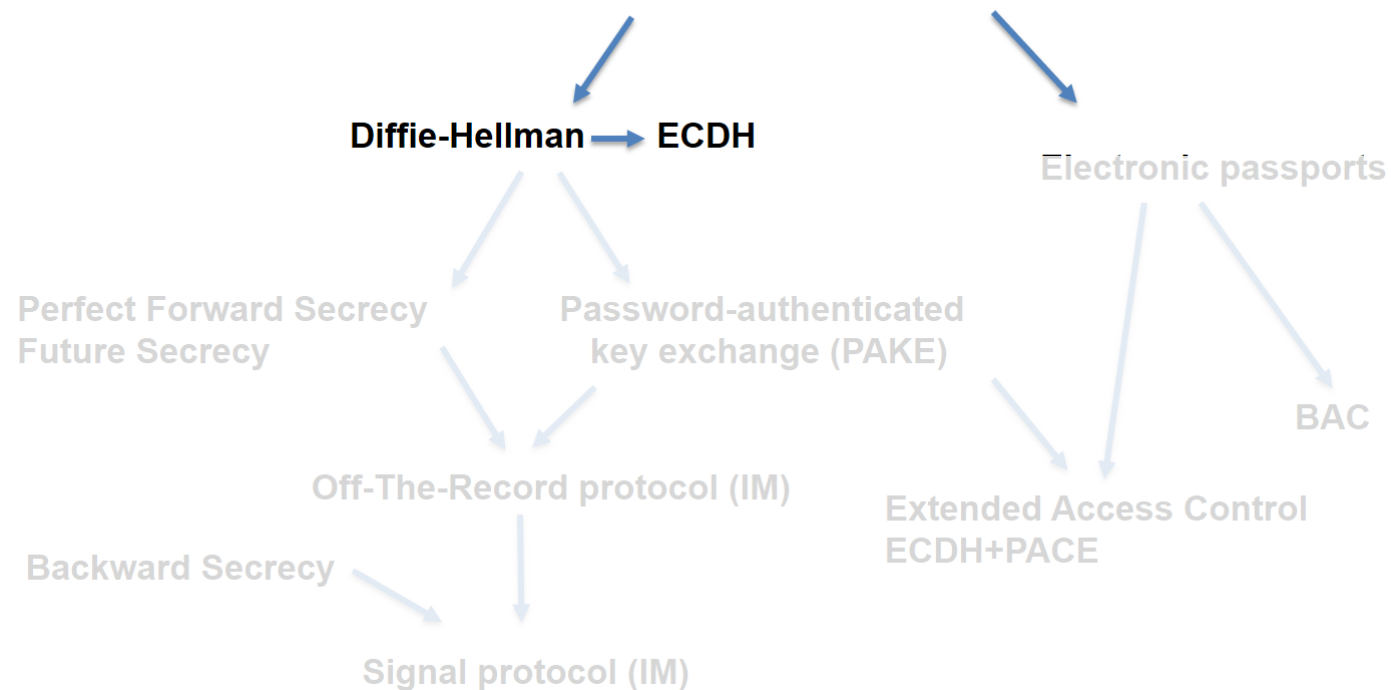
- Security Protocols Open Repository
  - <http://www.lsv.ens-cachan.fr/Software/spore/>
- C. Cremer, Scyther tool
  - <https://github.com/cascremers/scyther/>
- Cas Cremer's exercise sheet
  - <https://www.cs.ox.ac.uk/people/cas.cremers/scyther/scyther-exercises.html>

## N-S asymmetric: Fix

- Fixed by addition of B's identity into second step
  1.  $A \rightarrow B: \{A, N_A\}PK_{(B)}$
  2.  $B \rightarrow A: \{B, N_A, N_B\}PK_{(A)}$
  3.  $A \rightarrow B: \{N_B\}PK_{(B)}$



## 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)
6. ...

## 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	$R_1 = \text{random}()$ $E_{K'}(R_1) \longrightarrow$ $\longleftarrow E_{K'}(R_1, R_2)$ $E_{K'}(R_2) \longrightarrow$	$\text{random}() = R_2$
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	$\longleftarrow E_K(\text{data}) \longrightarrow$	

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  $\mathbf{p}$  and  $\mathbf{g}$ 
  - If  $\mathbf{g}$  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  $\mathbf{g}$  and  $\mathbf{p}$   $\Rightarrow$  fast discrete log to obtain  $A$
  - If  $\mathbf{p}$  is really prime and  $\mathbf{g}$  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)

## Diffie-Hellman Key Exchange

Step	Alice	Bob
1	Parameters: <b>EC curve, G (base point)</b>	
2	$A = \text{random}()$ $a = \mathbf{A \times G}$ (scalar multiplication)	$\text{random}() = B$ $\mathbf{B \times G} = b$
3	$a \longrightarrow$ $\longleftarrow b$	
4	$K = \mathbf{A \times B \times G} = \mathbf{A \times b}$	$\mathbf{B \times a} = \mathbf{A \times B \times G} = K$
5	$\longleftarrow E_K(\text{data}) \longrightarrow$	

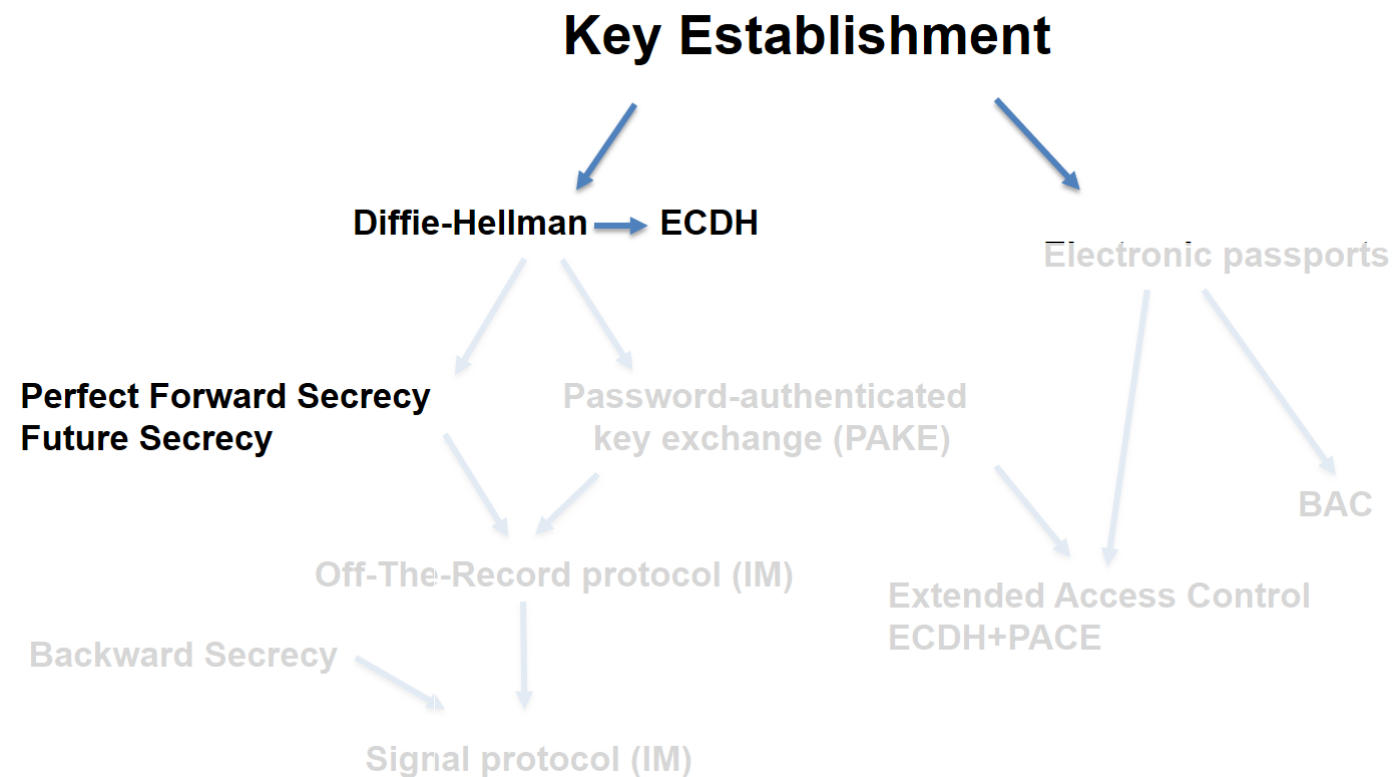
EC curve options:

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

<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  $K' = \text{KDF}(K)$  is used
  1. Original K may have weak bits
  2. Multiple keys may be required ( $K_{\text{ENC}}$ ,  $K_{\text{MAC}}$ )
- 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

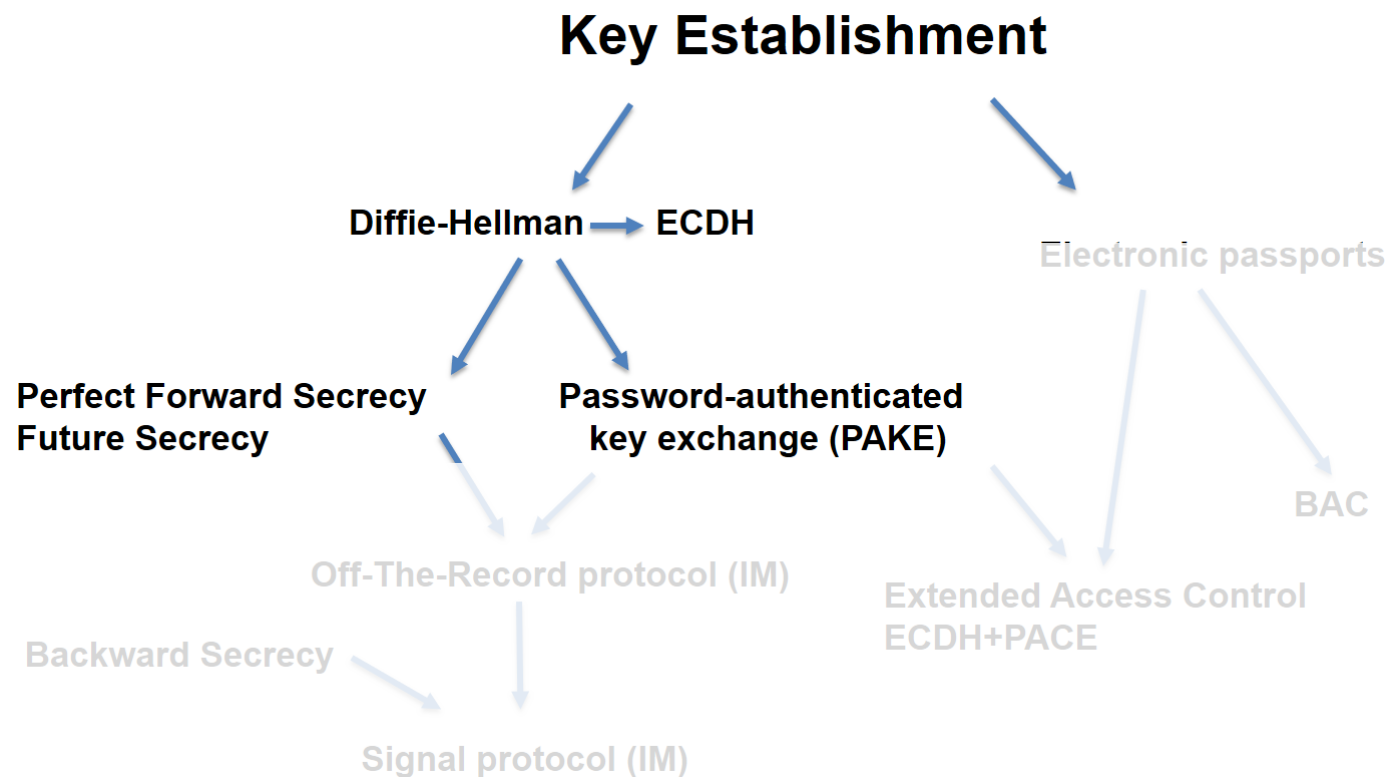
Must not derive future keys deterministically

## 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 can 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: $S = H(\text{password})$	
2	Parameters: $p, g$	
3	$A = \text{random}()$ $a = g^A \pmod p$	$\text{random}() = B$ $g^B \pmod p = b$
4a	$E_S(a) \rightarrow$ $\leftarrow E_S(b)$	
4b	$a \rightarrow$ $\leftarrow E_S(b)$	
4c	$E_S(a) \rightarrow$ $\leftarrow b$	
5	$K = g^{BA} \pmod p = b^A \pmod p$	$a^B \pmod p = g^{AB} \pmod p = K$
6	$\leftarrow E_K(\text{data}) \rightarrow$	

Various options  
a,b,c available

To protect against offline brute-force 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	$G = H(\text{password})^2$	$H(\text{password})^2 = G$
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	$\longleftarrow E_K(\text{data}) \longrightarrow$	

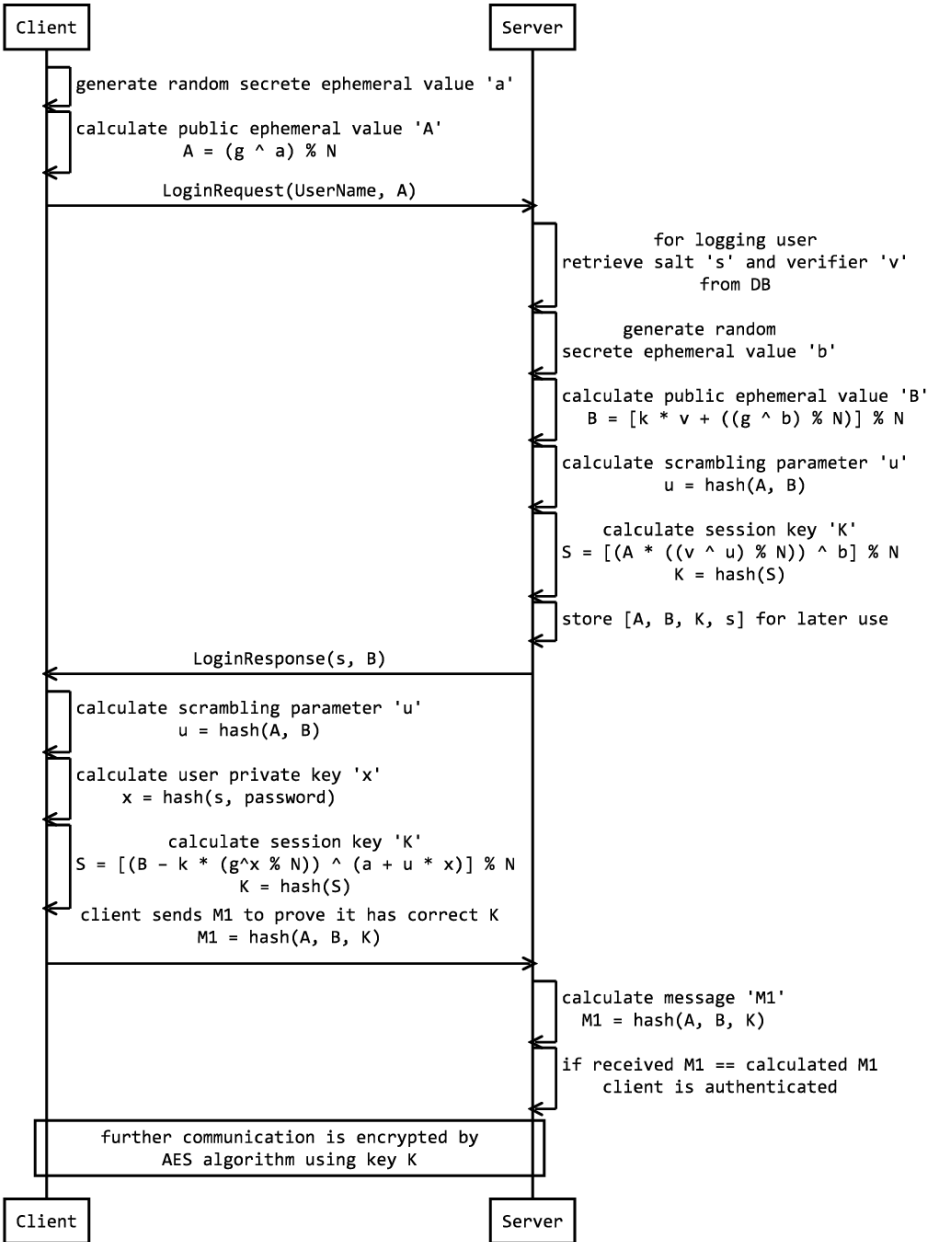
<http://www.themccallums.org/nathaniel/2014/10/27/authenticated-key-exchange-with-speke-or-dh-eke/>



## Secure Remote Password protocol (SRP), [aPAKE]

- Earlier Password-Authenticated Key Exchange protocols (PAKEs) were patented: EKE, SPEKE... (patent expired in 2017)
- Secure Remote Password protocol (SRP) 1998
  - Designed to work around existing patents
  - Royalty free, open license (Stanford 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  
 $N = \text{large safe prime number}$   
 $g = 2$   
 $k = \text{hash}(N, g)$



SRP is unnecessarily complex to work around existing patents

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

## PAKEs evolution

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

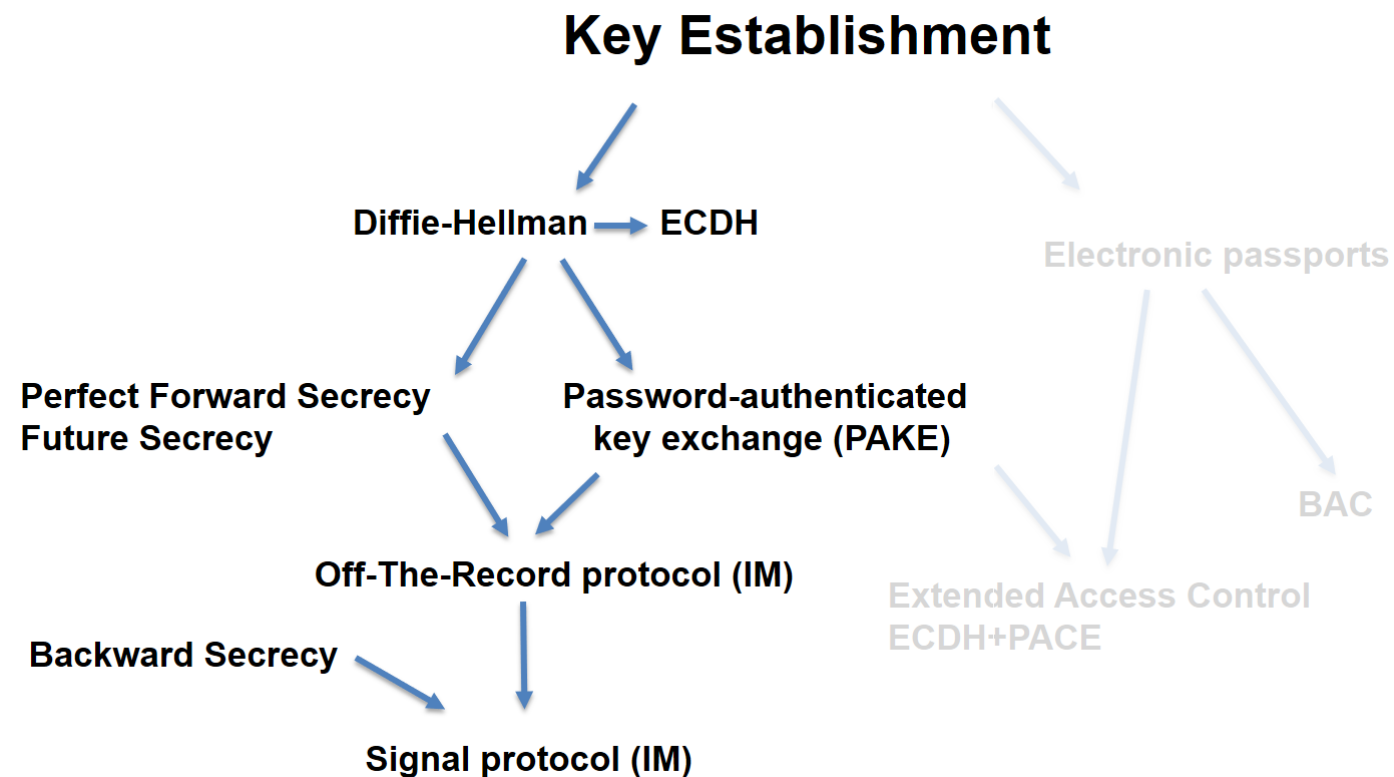
## Properties

- Compromised if server hack
- Prevent MitM offline cracking, still server hack compromise
- Like PAKE, but using salted hash instead of password, salt-specific precomputation possible
- Prevent offline cracking and precomputation 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>

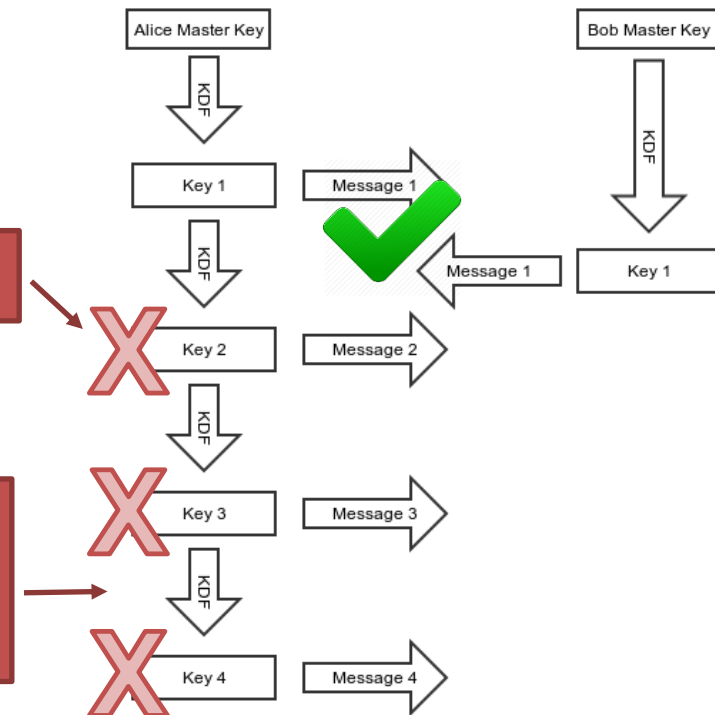


# SECURE INSTANT MESSAGING

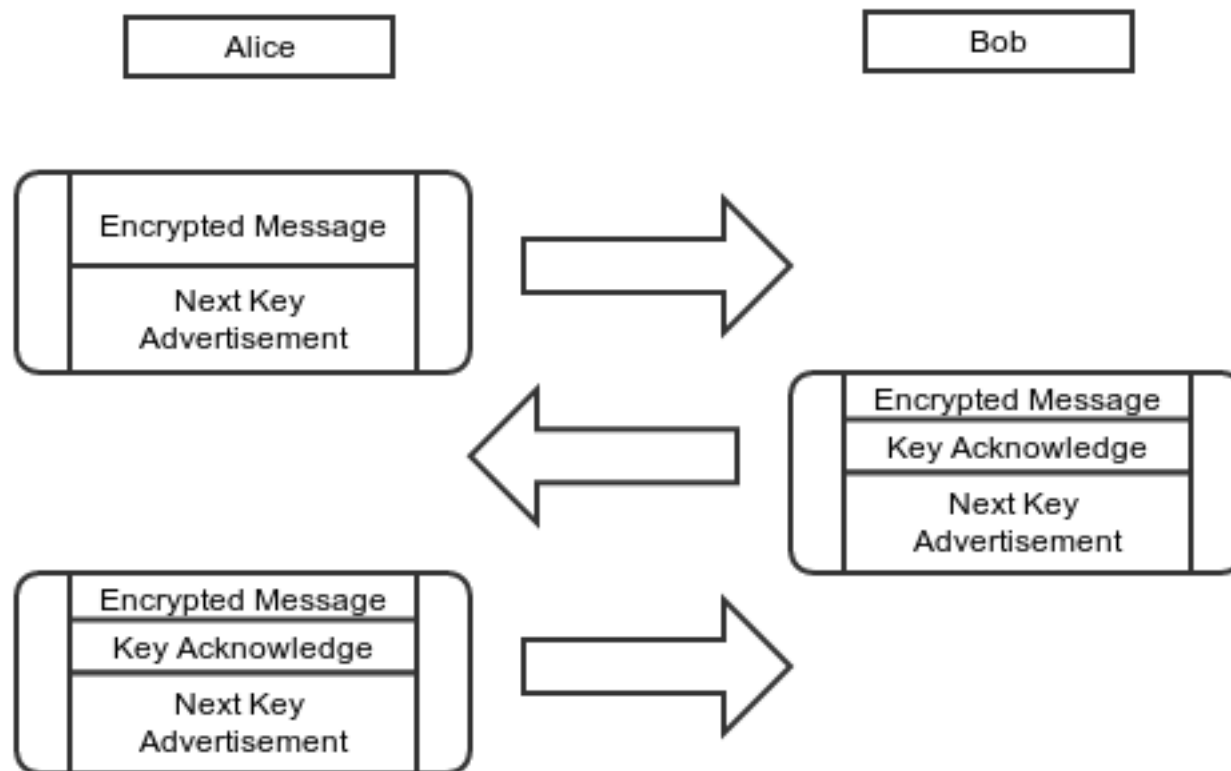
# “Toy” protocol for protection of instant messaging

<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)

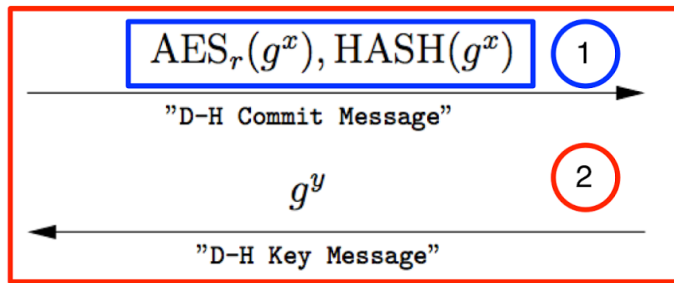


# OTR protocol

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

## Key exchange

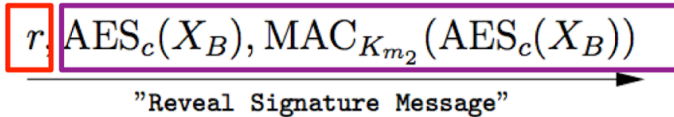
BOB ALICE



- 1. Hash commitment
- 2. Diffie-Hellman Key Exchange
- 3. Encrypted exchange of long-term keys & signatures

$$M_B = \text{MAC}_{K_{m_1}}(g^x, g^y, \text{pub}_B, \text{keyid}_B)$$

$$X_B = \{\text{pub}_B, \text{sig}_B(M_B)\}$$

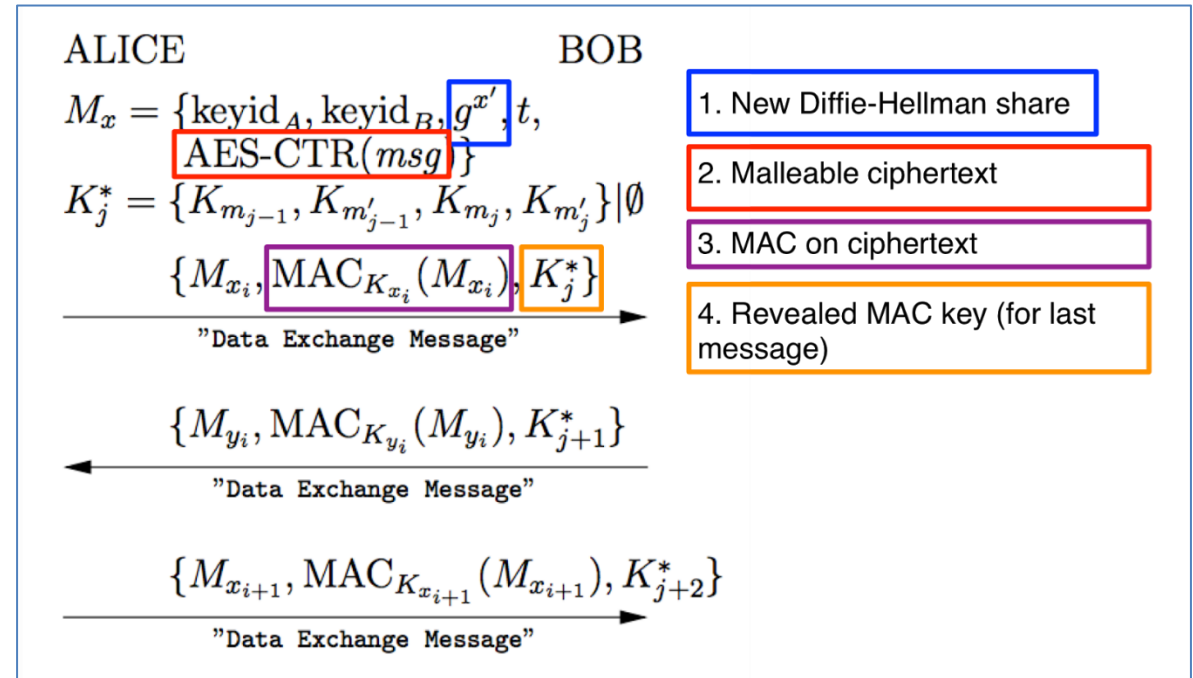


$$M_A = \text{MAC}_{K_{m'_1}}(g^y, g^x, \text{pub}_A, \text{keyid}_A)$$

$$X_A = \{\text{pub}_{A'}, \text{keyid}_{A'}, \text{sig}_A(M_A)\}$$



## Message exchange



- 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

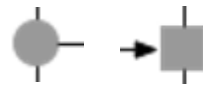
- State-of-the-art of instant messaging protocols
  - Used in Signal, WhatsApp, Facebook Messenger, Google Allo...
- The Signal protocol provides:
  - confidentiality, integrity, message authentication,
  - participant consistency, destination validation,
  - forward secrecy, backward secrecy (aka future 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



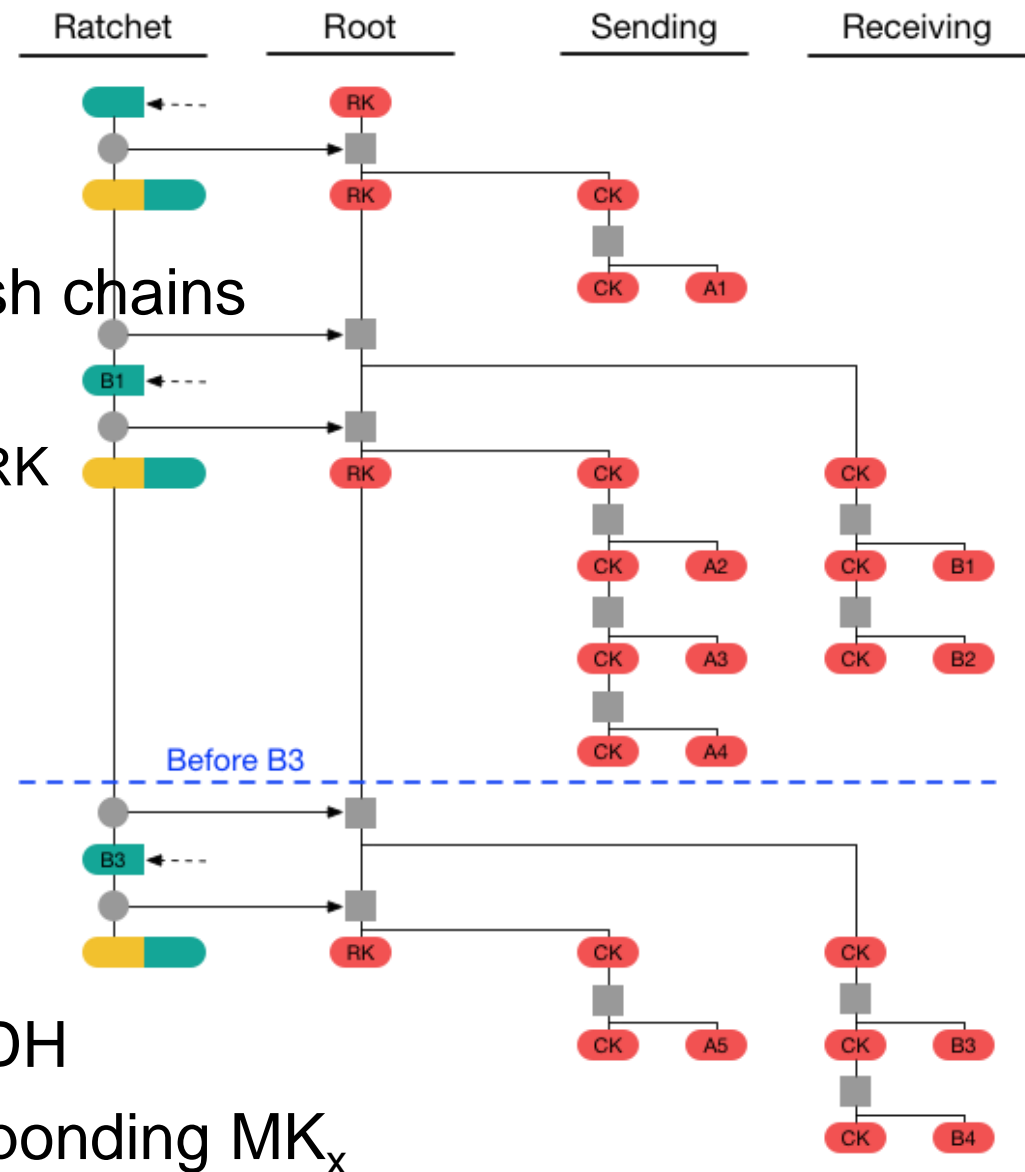
# The Signal protocol implementation

- 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 hash chains
- Root key(s) (RK)
  - Established from last ECDH ratchet and previous RK
- Chain key(s) (CK)
  - Established from the most recent RK + hash chain
  - KDF to derive next CK =  $\text{HMAC-HASH}(\text{CK}, "1")$
- Message key(s) (MK)
  - Derived from CK as  $\text{MK} = \text{HMAC-HASH}(\text{CKs}, "0")$
  - Message  $A_x$  encrypted by  $\text{MK}_x$
- RK&CK compromise is “healed” by next ECDH
- Out-of-order messages by storage of corresponding  $\text{MK}_x$





Get Signal Support



## The Double Ratchet Algorithm

Revision 1, 2016-11-20 [PDF]

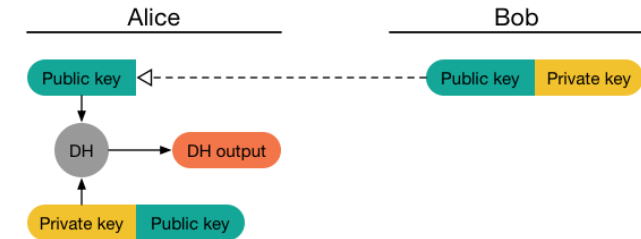
Trevor Perrin (editor), Moxie Marlinspike

### Table of Contents

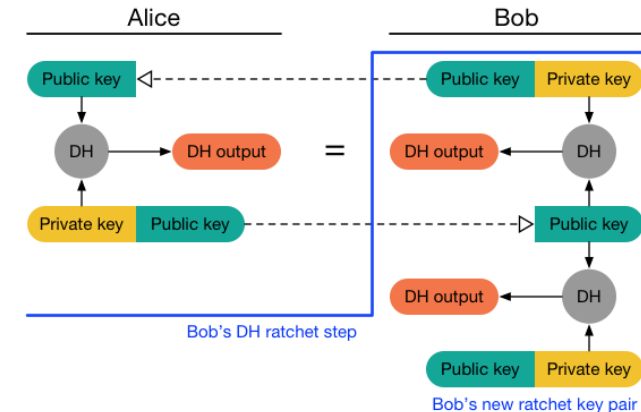
- 1. Introduction
- 2. Overview
  - 2.1. KDF chains
  - 2.2. Symmetric-key ratchet
  - 2.3. Diffie-Hellman ratchet
  - 2.4. Double Ratchet
  - 2.6. Out-of-order messages
- 3. Double Ratchet
  - 3.1. External functions
  - 3.2. State variables
  - 3.3. Initialization
  - 3.4. Encrypting messages
  - 3.5. Decrypting messages
- 4. Double Ratchet with header encryption
  - 4.1. Overview
  - 4.2. External functions
  - 4.3. State variables
  - 4.4. Initialization
  - 4.5. Encrypting messages
  - 4.6. Decrypting messages
- 5. Implementation considerations
  - 5.1. Integration with X3DH
  - 5.2. Recommended cryptographic algorithms
- 6. Security considerations
  - 6.1. Secure deletion
  - 6.2. Recovery from compromise
  - 6.3. Cryptanalysis and ratchet public keys
  - 6.4. Deletion of skipped message keys
  - 6.5. Deferring new ratchet key generation
  - 6.6. Truncating authentication tags
  - 6.7. Implementation fingerprinting
- 7. IPR
- 8. Acknowledgements
- 9. References



Get Signal Support Blog Developers Career



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:



## The Signal protocol – group messaging

- Speaker consistency
- Out-of-order resilience
- Dropped message resilience
- Computational equality
- Trust equality
- Subgroup messaging
- Contractible and expandable group membership
- Read more details at <https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf>

*[https://en.wikipedia.org/wiki/Signal\\_\(software\)#Encryption\\_protocols](https://en.wikipedia.org/wiki/Signal_(software)#Encryption_protocols)*



# 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 Lightning 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

# Some existing Noise patterns

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

**NN:**

-> e

<- e, ee

**NX:**

-> e

<- e, ee, s, es

- 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
  - ...

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/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

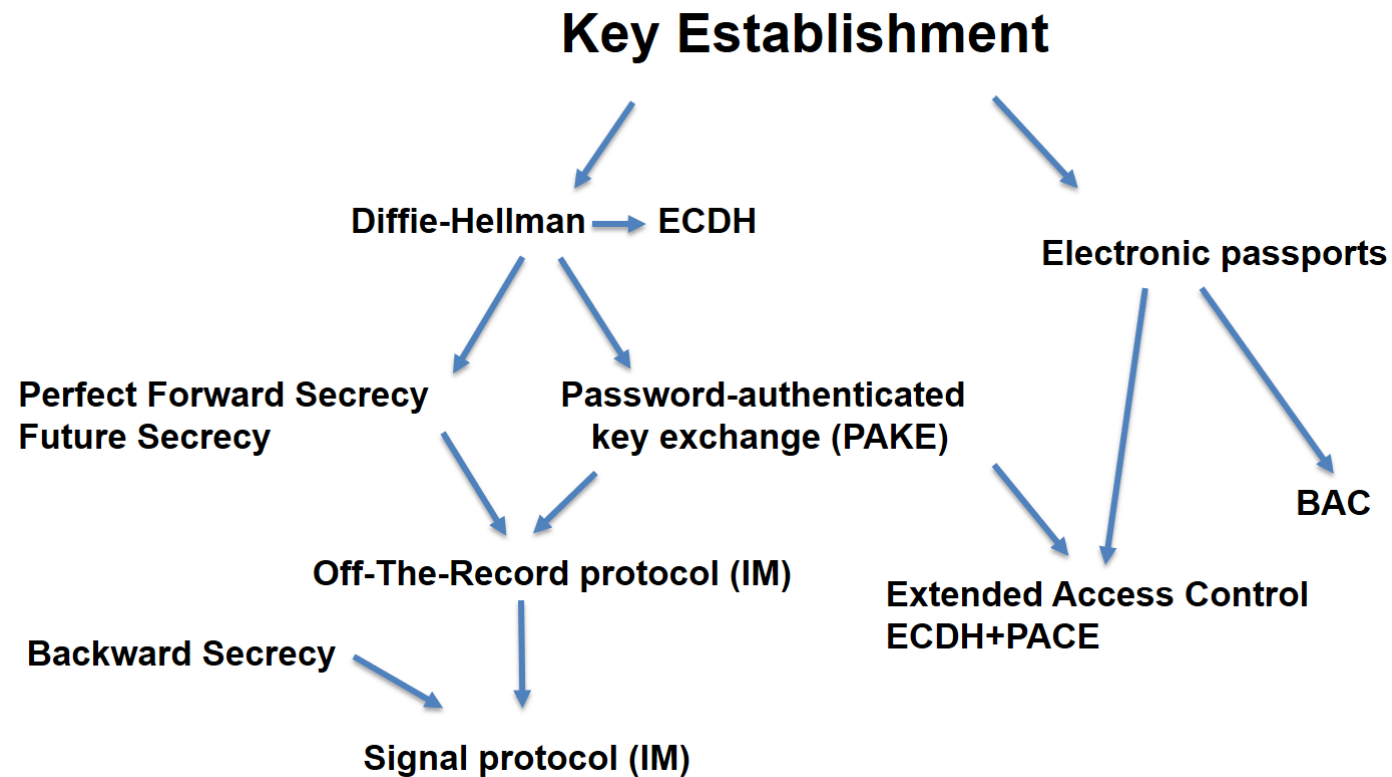
Get Implementation

WRITTEN IN RUST

Generate Rust Implementation Code for WebAssembly Builds

Get Implementation

WRITTEN FOR WASM

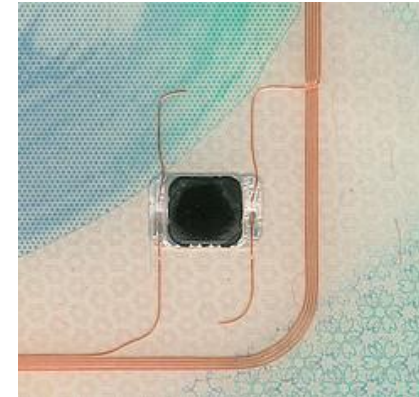


# ELECTRONIC PASSPORTS AND CITIZEN ID CARDS

*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)





## Data groups

Data group	Stored data
DG1	<b>Machine readable zone (MRZ)</b>
DG2	<b>Biometric data: face</b>
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]

# Basic Access Control (BAC) protocol

- Authentication & secure channel between inspection terminal and chip
  - Based on symmetric crypto (3DES), similar to SCP'0x protocols
  - Low computational requirements
- Problem: anyone with access to MRZ can authenticate
- Problem: MRZ has insufficient entropy
  - Document number, birth date, expiration date used
  - Theoretically 58/74 bits, but in practice about 32 bits
- Offline attack (eavesdrop then crack)
  - Eavesdrop valid communication between chip and reader
  - Brute-force attack in less than hour ( $2^{32}$  ops, offline attack)
- Online attacks against chip (att. model: found passport)
  - Significantly slower, ~20 ms for every attempt

# Extended Access Control (EAC) protocol

- Based on asymmetric cryptography (RSA/DH/ECDSA)
- Chip Authentication (CA) based on PACE protocol
  - Password Authenticated Connection Establishment (PACE)
  - Uses chip's static DH/ECDSA key and terminal's ephemeral DH key pair (perfect forward secrecy)
  - Both parties combines chip's public static and ephemeral public key into same key  $K$
  - Keys for encryption and MAC ( $K_{ENC}$ ,  $K_{MAC}$ ) are derived from exchanged  $K$
- How can be Terminal sure of authenticity of chip's static key?
  - Signed by Issuing country
- Terminal Authentication (TA)
  - Based on challenge-response protocol (RSA/ECDSA, SHA-1/2)
  - Hash of the ephemeral DH key from previous step hashed with challenges

## 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/>





# DESIGN OF PROTOCOLS

# Design of cryptographic protocols

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

## Design principles I. (Abadi & Needham)

- The conditions for a message to be acted should be clearly set out so reviewer can judge if they are acceptable.
  - Documentation, diagrams, formal specification
- Every message should say what it means, message interpretation should depend only on its content.
  - “This is 2<sup>nd</sup> message of SCP’02 from A to B”
  - No assumptions like next random chunk number should be encrypted 2<sup>nd</sup> message because I just received 1<sup>st</sup> message
- Mention name of principal (“Alice01”)
  - Prevents (if checked) unintended parallel runs of protocol
  - Prevents reflection attack

## Design principles II. (Abadi & Needham)

- Be clear about why encryption is being done
  - For confidentiality, not to “somewhat” ensure integrity
- When signing encrypted data, it should not be inferred that signing entity knows data content
  - No knowledge of encryption key
- Be clear about properties of nonce
  - random, never repeated, unpredictable, secret
  - Random → almost never repeated unintentionally

## Design principles III. (Abadi & Needham)

- If predictable quantity is to be effective, it should be protected so that an intruder cannot simulate a challenge and later replay the message
  - Counter as challenge → counter freshness verification necessary → state
- If timestamps are used as freshness guarantees, then difference between local clocks at various machines must be much less than allowable age of message
  - Otherwise an attacker can replay within time window
- Key may have been used recently and yet be old and possibly compromised
  - Clear session state after session end, check freshness

## Design principles IV. (Abadi & Needham)

- It should be possible to deduce which protocol and which run of that protocol a message belongs to including order number in the protocol
  - Danger of parallel runs of same protocol
  - MAC and chaining with fresh session keys prevents message mixing
- Trust relation should be made explicit and there should be good reason for its necessity.
  - Less trust needed → better security achieved

## Design principles V. (Hanno Böck)

- Always use an AEAD. No CBC, OFB, CFB. No "signatures are as good as an AEAD".
- Stay away from PKCS #1 1.5. If you want to use RSA use PSS/OAEP, but maybe don't use RSA.
- Don't use ECDSA, don't use any old ECC. Use X25519, Ed25519 or alike.
- Don't use DSA, 64-bit-blocks, sha1/md5 and other old crap.
- Think about duplicate nonces. If you can't easily avoid nonce repetition consider AES-SIV.
- *Still talk to a real cryptographer, but if you follow these you're already better than a lot of others :-)*



## Secure channel – typical composition

1. Exchange basic (public) parameters
2. Generate random challenges (freshness)
3. Use pre-distributed secrets and challenges to generate session keys (protect long term secrets)
4. Compute and verify authentication cryptograms (entity authentication)
5. Encrypted&MAC message(s) (Secure Messaging)
6. End secure channel (erase session keys)

## SCP – what to take into account

- Usage scenario and expected attackers
- Confidentiality and integrity of command data
- Network level attacks (replay...)
- Atomicity of critical operations
- Robustness against side channel attacks (time and power analysis, fault attacks...)
- Robustness against incorrect attempts (limit, delay retries...)
- Resilience against traffic analysis
- API and implementation attacks

## SCP – usage scenario and attacker model

- What are the sensitive objects (keys, data, functions)?
- What are these sensitive objects used for and what is the data flow of these objects?
- What are the capabilities of the attackers (funding, tools, knowledge)?
- What are the points where an attacker can observe the system (dump of exchanged messages, debugging, ...)?
- Which parts of the system must be trusted to achieve required functionality (less the better)?

## SCP – network attacks

- Use HMAC or OMAC instead of simple hash only
- Include command header/metadata into MAC
- Pre-share two keys (encryption, mac) or derive from master instead of using only one
- Use pre-shared keys only to derive session keys. Session keys are used than to generate cryptograms etc.
- Session keys must be dependent on contributions from both parties. One party cannot force resulting key into specific value

## SC – network attacks

- Replay attack – hash chain better than counter only
- Encrypt then MAC:  $\text{MAC}(\text{ENC}(\text{data}))$
- Close channel on error
- Use GCM rather than CBC rather than ECB
- Be aware of block swap in ECB mode, cut attack in CBC
- Do not use XOR for combination of values – use hash/HMAC instead
- Reflection attack: Do not use symmetric protocol messages ( $A \rightarrow B$  cannot be reflected as  $B \rightarrow A$ )

# **ELECTRONIC PASSPORTS - MORE DETAILS**

## Authorization and passports

1. Inspection terminal to read basic info from chip
2. Inspection terminal to read biometric data from chip
3. You to enter country based on chip data

# Basic Access Control (BAC) protocol

- Authentication & secure channel between inspection terminal and chip
  - Based on symmetric crypto (3DES), similar to SCP'0x protocols
  - Low computational requirements
- Problem: anyone with access to MRZ can authenticate
- Problem: MRZ has insufficient entropy
  - Document number, birth date, expiration date used
  - Theoretically 58/74 bits, but in practice about 32 bits
- Offline attack (eavesdrop then crack)
  - Eavesdrop valid communication between chip and reader
  - Brute-force attack in less than hour ( $2^{32}$  ops, offline attack)
- Online attacks against chip (att. model: found passport)
  - Significantly slower, ~20 ms for every attempt

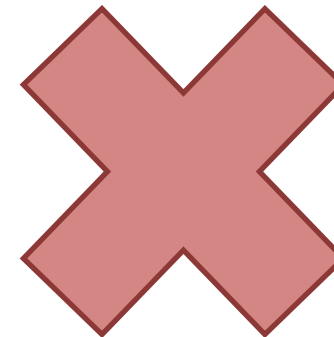


## EAC – motivation

- EU passports stores fingerprints (from 2009)
  - More sensitive than facial photo => better protocol needed
- Goal: not everyone with access to passport (and MRZ for BAC) should be able to read out fingerprint
  - Issuing country decides who else can access
- Stronger authentication than BAC required

## Mind exercise: symmetric crypto

- What if only symmetric crypto is used?
  - Every chip has own unique symmetric key
  - Large number of keys in inspection terminals
  - Compromise of single terminal breach security
  - => impractical and insecure => not used



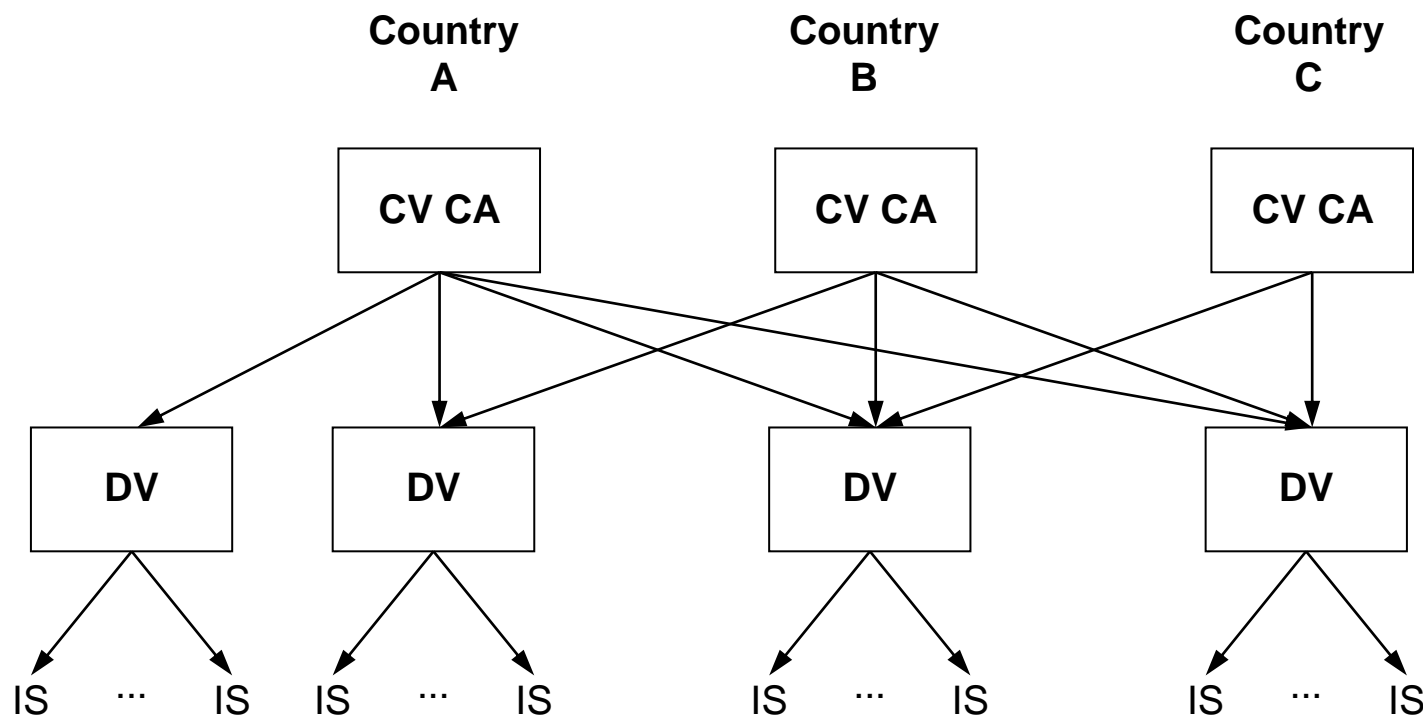
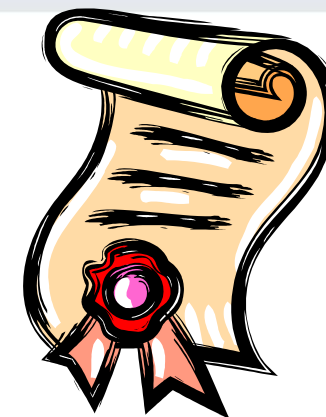
# Extended Access Control (EAC) protocol

- Based on asymmetric cryptography (RSA/DH/ECDSA)
- Chip Authentication (CA) based on PACE protocol
  - Password Authenticated Connection Establishment (PACE)
  - Uses chip's static DH/ECDSA key and terminal's ephemeral DH key pair (perfect forward secrecy)
  - Both parties combines chip's public static and ephemeral public key into same key  $K$
  - Keys for encryption and MAC ( $K_{ENC}$ ,  $K_{MAC}$ ) are derived from exchanged  $K$
- How can be Terminal sure of authenticity of chip's static key?
  - Signed by Issuing country
- Terminal Authentication (TA)
  - Based on challenge-response protocol (RSA/ECDSA, SHA-1/2)
  - Hash of the ephemeral DH key from previous step hashed with challenges

## Terminal authentication I.

- Only authorized border authorities can read the secondary biometric data (fingerprints and iris)
  - The inspection system must prove to the chip it is authorized
  - The chip stores a trust point – root certificate
  - Inspection system presents a valid certificate chain (starting from the passport's trust point) specifying the IS's authorizations (e.g. to read DG3)
  - Challenge-response where IS proves knowledge/access of a secret key (whose public part is certified)
  - Certificates in Card-verifiable (CV) format

# Terminal Authentication II



Root public key of issuing country: CV CA

## Terminal Authentication III

- Supported algorithms
  - RSA PKCS#1 v1.5 with SHA-1 or SHA-256
  - RSA PSS with SHA-1 or SHA-256
  - ECDSA with SHA-1, SHA-224 or SHA-256
  - Key lengths
    - For ECDSA allowed 160, 192, 224, 256 bits
    - For RSA allowed 1024, 1280, 1536, 2048 and 3072 bits
    - In practice ECDSA is more common, key lengths 192 and 224 bit most popular, existing implementations also support 256 and 384 bits.
    - For RSA the PKCS#1 v1.5 padding is much more popular than PSS, key lengths are between 512 (test only) and 2048 bits.

# Active Authentication (AA) protocol

- Motivation: Prevent cloning of passport
  - Is chip inside passport authentic?
- Passport-specific asymmetric key stored on chip
- Public key freely readable (DG15 file, hash signed)
- Authentication against terminal
  - Terminal generates 8 random bytes
  - Chip adds additional 8 random bytes, hash and sign
  - Terminal verifies signature
- Privacy attack: terminal's challenge is date → signed
- PACE protocol replaces Active Authentication

# EAC Chip authentication

- Verifies the authenticity of the chip
  - Replaces Active Authentication
  - Improves the security of Secure Messaging
    - Keys with higher entropy than for Basic Access Control (BAC)
- Diffie-Hellman key pair (passport's chip)
  - Public key and domain parameters stored in DG14
  - Private key never leaves the chip
- Diffie-Hellman key pair (inspection system – IS)
  - Key pair generated according to passport's domain parameters
  - Private key kept secret, public key sent to chip
- Both the parties (passport, IS) now share a secret
  - The secret is used to derive the SM keys, the ability to communicate proves the authenticity of the chip

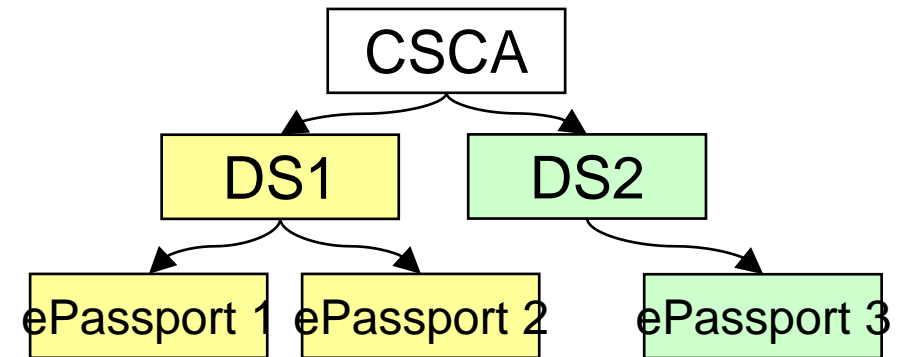


## Chip authentication II

- Algorithms supported by the EAC specification
  - Diffie-Hellman (PKCS#3)
  - Elliptic Curve Diffie-Hellman (ISO 15946, BSI TR-03111 )
  - Both DH and ECDH popular with chip manufacturers
    - DH with 1024 or 1536 bit prime
    - ECDH with mostly 224 bit curves, some implementations use 256 or 384 bits
- No challenge semantics
- Coexistence of CA and AA

# Passive Authentication

- Goal: are data in chip unchanged?
- The list of the hashes (SHA-1/2) of all present data groups is digitally signed by the issuing organisation (Document Signer)
  - State printing house, Embassy, Etc.
- The X.509 certificate of the Document Signer issued by the CA of the issuing country (Country Signing CA – e.g. the ministry of interior) is included.
- The CSCA certificates must be exchanged bilaterally
- ICAO PKD for DS certificates, CSCA CRL and cross certificates
- Passive authentication is a **mandatory** security feature of all ePassports



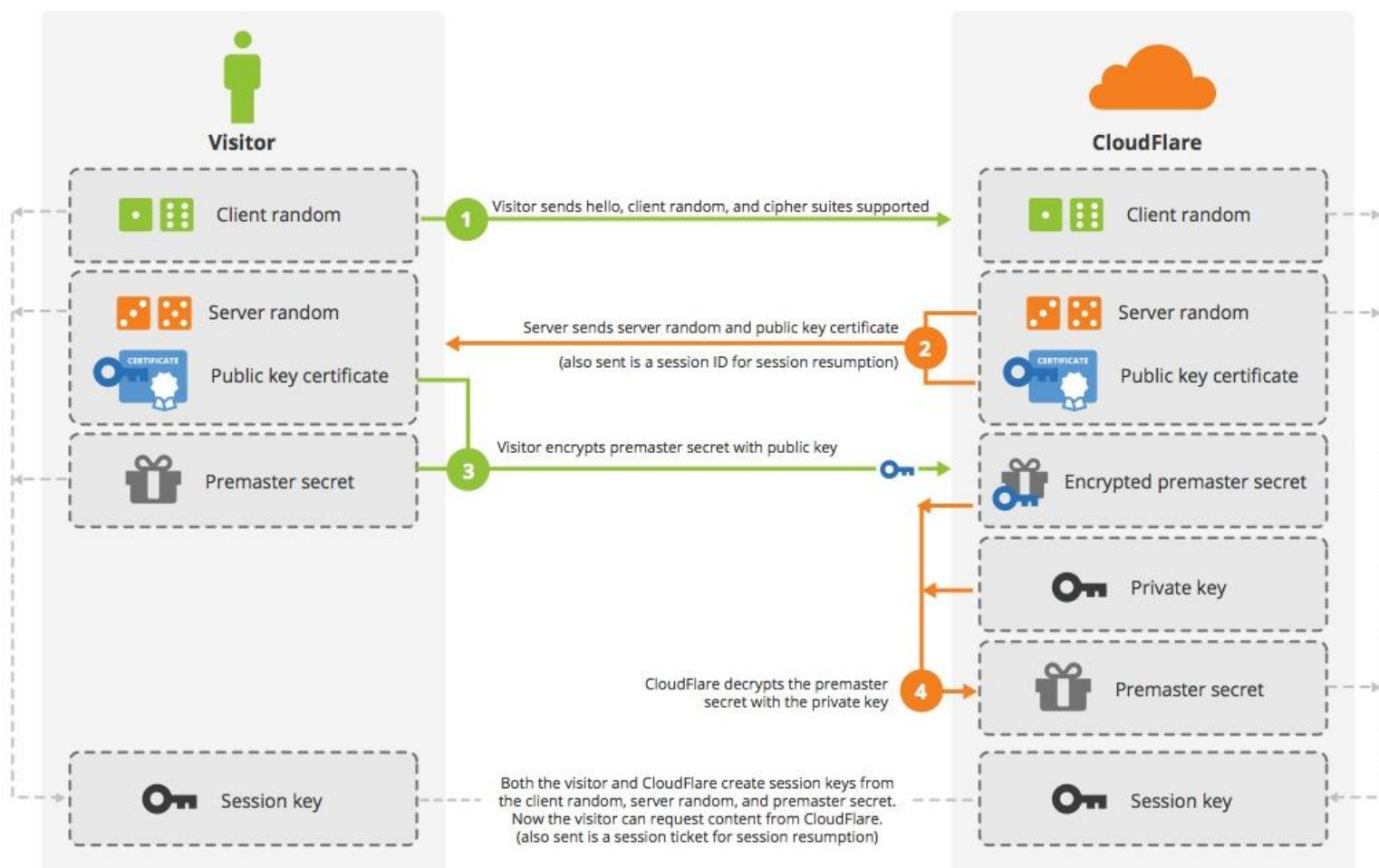
# Passive Authentication – the details

- The file EF.SOD contains a CMS (PKCS#7) SignedData structure (file is read and validated by inspection system)
  - The signed data is the list of hashes of the present data groups
  - The DS certificate is included (ICAO optional, EU mandatory)
  - Data is signed by the DS
- Signature schemes
  - RSA with PKCS#1 v1.5 padding (min. 3072 bits for CSCA, 2048 bits for DS)
    - E.g. Hungary, France, Spain, Portugal, Italy: RSA 4096 bit key with SHA-1
    - E.g. Austria, Netherlands: RSA 3072 bit key with SHA-256
  - RSA with PSS padding (min 3072 bits for CSCA, 2048 for DS)
    - E.g. Czech Republic, Norway, Denmark, Japan and Australia – all with SHA-256
  - DSA (not standardized for key lengths > 1024)
  - ECDSA (domain parameters must be specified, min. 256 bits for CSCA, 224 bits for DS)
    - E.g. Switzerland, Germany, Russia – all with SHA-1
- Message Digest algorithms (for signature schemes and hashes of Data Groups)
  - SHA-1 and all SHA-2 algorithms

How to authenticate and communicate securely?

# SECURE CHANNEL PROTOCOL (FOR SMARTCARDS)

# TLS handshake



*Credit: Cloudflare*

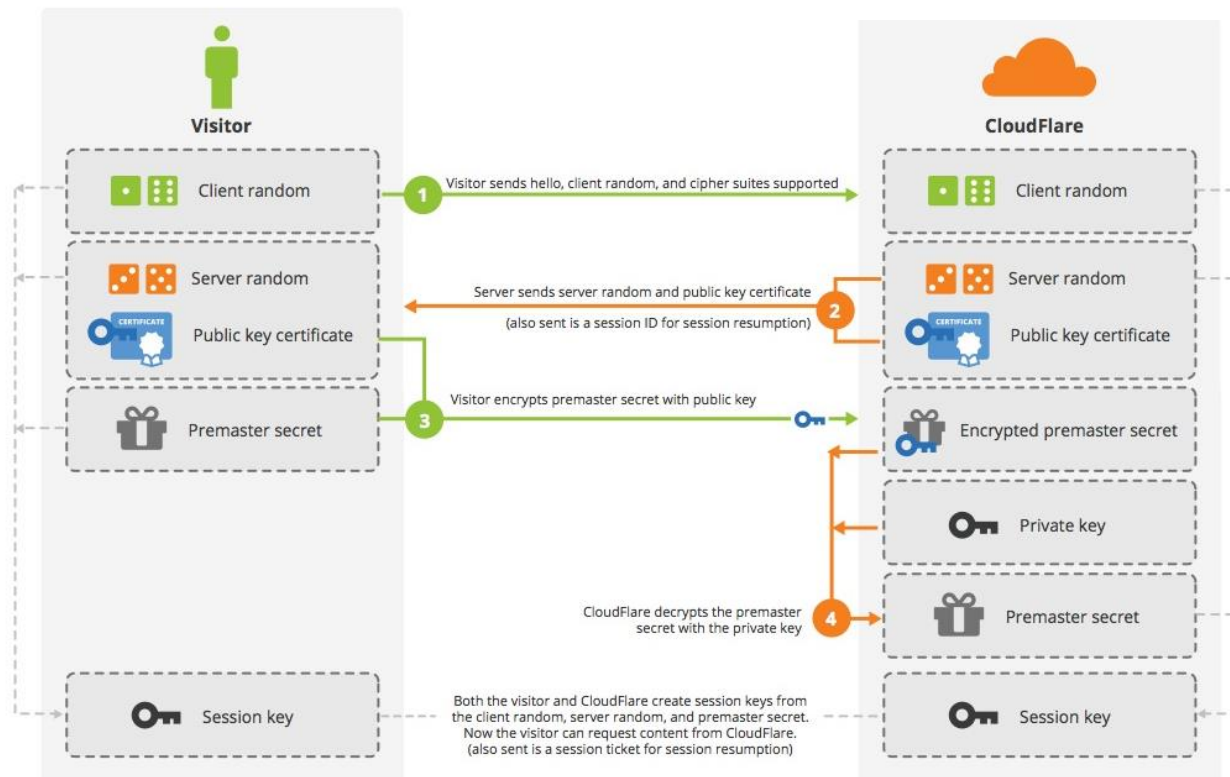
# Why not to use TLS all the time?

1. Requires asymmetric cryptography
    - Unsuitable for slower devices
  2. Requires long keys
    - Unsuitable for devices with small memory
  3. Requires significant data overhead (~6.5KB)
    - <http://netsekure.org/2010/03/tls-overhead/>
  4. More lightweight protocols exist
    - RFID / smartcards / IoT...
- Note: TLS can be fully implemented on smartcards (but slow)
    - [https://github.com/gilb/smart\\_card\\_TLS](https://github.com/gilb/smart_card_TLS)

## Secure channels – questions to ask

- What attacker model is assumed?
- Integrity protection? Encryption? Authentication?
- One-side or mutual authentication?
- What kind of cryptography is used?
- What keys are required/pre-distributed?
- Additional trust hierarchy required?
- Is necessary to generate random numbers/keys?
- What if keys are compromised? Forward secrecy?

- What attacker model is assumed?
- Integrity protection? Encryption? Authentication?
- One-side or mutual authentication?
- What kind of cryptography is used?
- What keys are required/pre-distributed?
- Additional trust hierarchy required?
- Is necessary to generate random numbers/keys?
- What if keys are compromised? Forward secrecy?





## Common lightweight SCPs

- OpenPlatform SCP'01,'02 (3DES-based)
- OpenPlatform SCP'10 (RSA-based)
- OpenPlatform SCP'03 (AES-based)
- ISO/IEC 7816-4 Secure Messaging
- ePassports Basic Access Control (3DES-based)
- ePassports Extended Access Control (3DES,RSA,DH,SHA1/2-based)

## Example: GlobalPlatform SCP'03

- Mutual authentication (based on symmetric crypto)
- Session key derivation (based on long-term keys)
  - NIST SP 800-108
- Message (APDU) confidentiality and integrity MAC

### 1. INITIALIZE UPDATE

- Random challenge, card's computations

### 2. EXTERNAL AUTHENTICATE

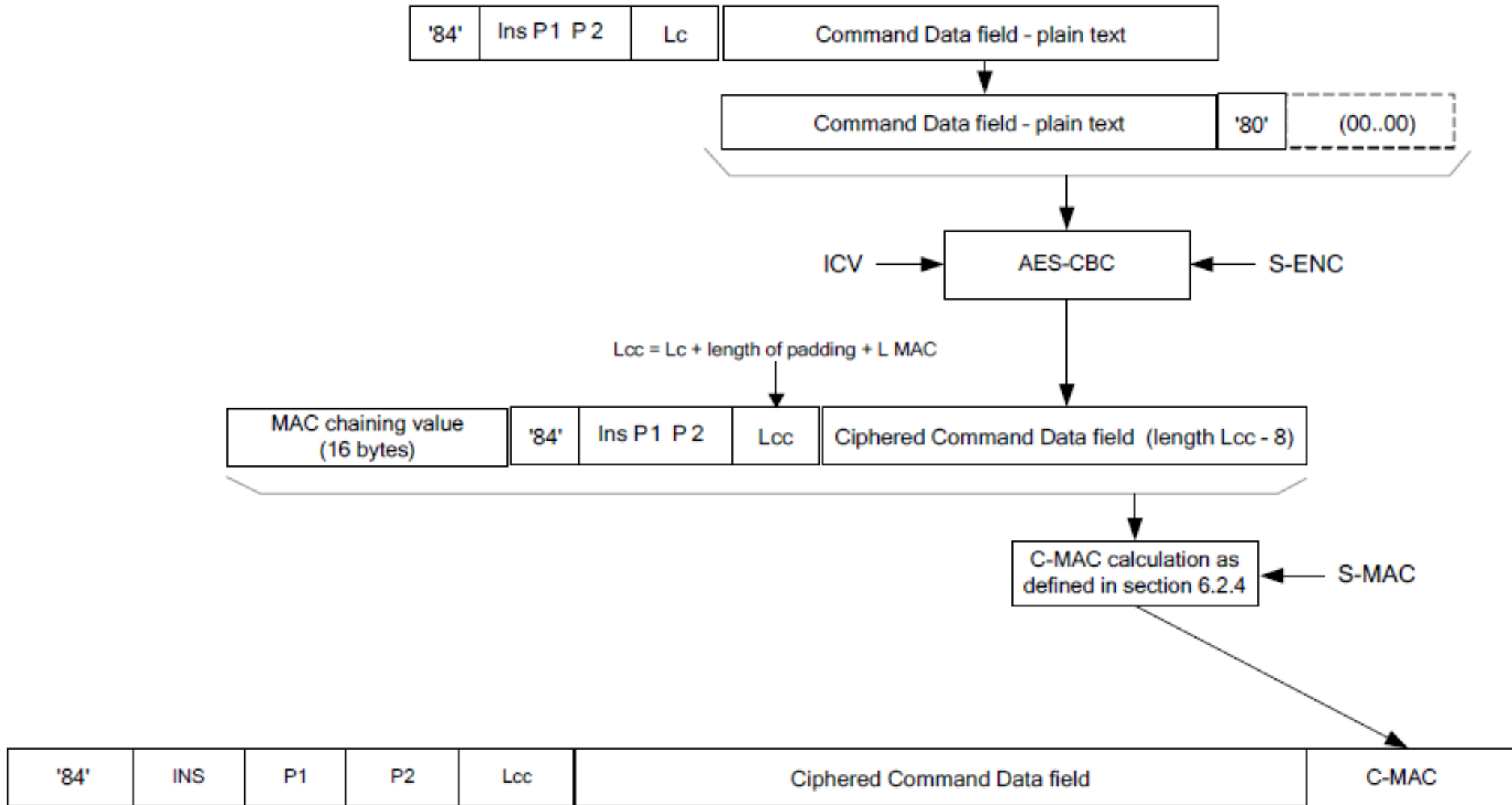
- Terminal response

### 3. Secure messaging

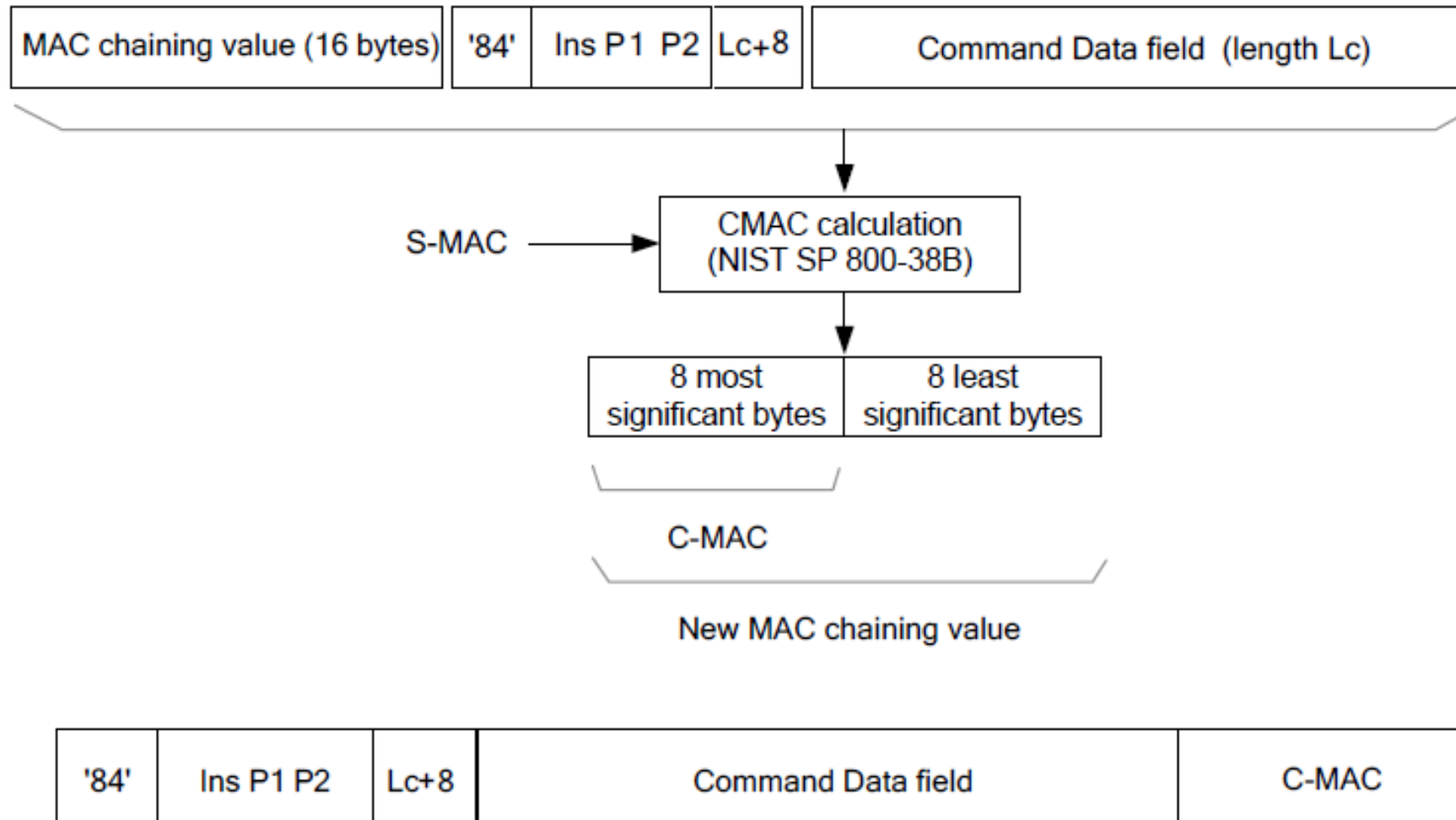


What are problems with usage of symmetric crypto?

Figure 6-4: APDU Command Data Field Encryption



**Figure 6-1: APDU C-MAC Generation**



*Secure Channel Protocol '03', Card Specification v2.2 – Amendment D, GPC\_SPE\_014*

## ePassport protocols (ICAO 9303)

- Significantly more complex trust model
  - Passport, Inspection terminal, Trusting countries, Distrusting countries
  - Multiple sensitivity levels (basic info / fingerprint / iris)
  - Combination of symmetric and asymmetric cryptography
- Basic Access Control (BAC) protocol
  - SCP-like protocol, static key is content from MRZ
- Extended Access Control (EAC) protocol
  - Terminal authentication (RSA/ECDSA, SHA-1/2)
  - Chip authentication (DH/ECDSA key)
  - PACE protocol to establish session keys
- Active Authentication (AA) protocol