Applied Cryptography
Spring Semester 2023

Lecture 35: The Signal Messaging Protocol

Felix Günther
(Slides by Benjamin Dowling)
Applied Cryptography Group
https://appliedcrypto.ethz.ch/

Outline

Secure Messaging and Signal

Signal Overview

The Signal Double Ratchet Protocol

The Signal X3DH Protocol

Extra slides: Signal's Message Encryption and Algorithms

Extra slides: Signal Authentication

Outline

Secure Messaging and Signal

Signal Overview

The Signal Double Ratchet Protocol

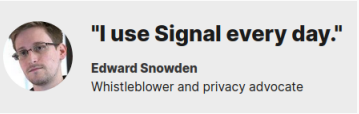
The Signal X3DH Protocol

Extra slides: Signal's Message Encryption and Algorithms

Extra slides: Signal Authentication

Promoted by Privacy Advocates

State-of-the-art end-to-end encryption (powered by the open source Signal Protocol) keeps your conversations secure. We can't read your messages or listen to your calls, and no one else can either. — Signal



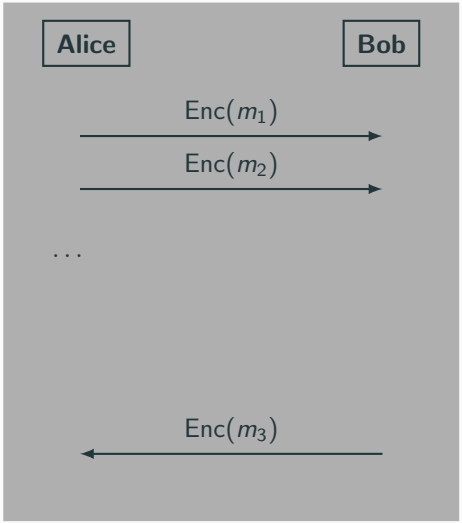
Interesting to Academics

- Two party asynchronous E2EE messaging protocol
- Explosive uptake:
  Signal, WhatsApp, Wire, Facebook Silent Messenger, and many more!
- Cool properties:
  Forward **S**ecurity
  Post-**C**ompromise **S**ecurity
- Complex design:
  X3DH: initial key exchange
  Double Ratchet: asymmetric and symmetric ratcheting...

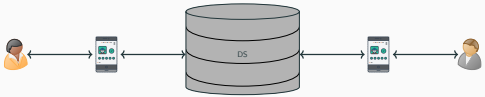
Two Party E2EE messaging protocol

Two party asynchronous E2EE messaging protocol:

- Two party messaging protocol - Alice and Bob get to communicate with each other
- E2EE - Only Alice and Bob able to read messages

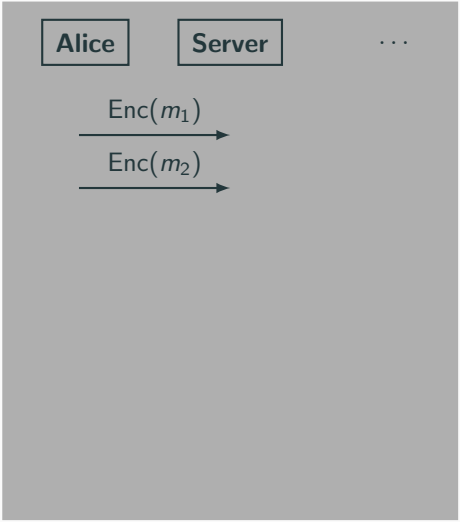


Asynchronous Protocol

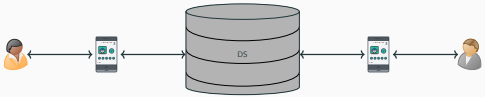


Asynchronous messaging protocol:

- Asynchronous - Parties not required to be online at the same time
- Server now needed to handle message delivery

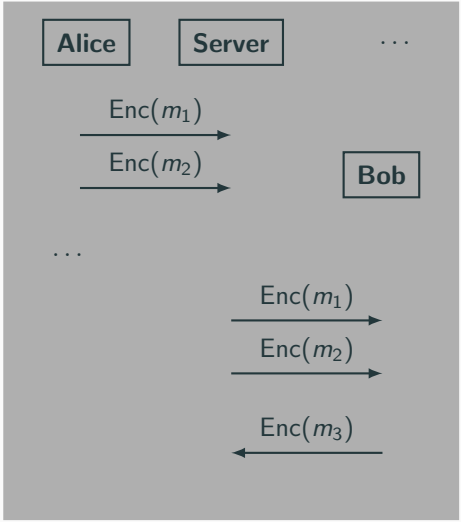


Asynchronous Protocol (2)

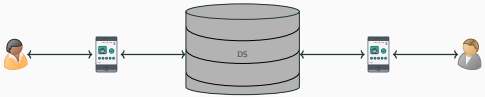


Asynchronous messaging protocol:

- Asynchronous - Parties not required to be online at the same time
- Server now needed to handle message delivery
- Server delivers messages when parties come online

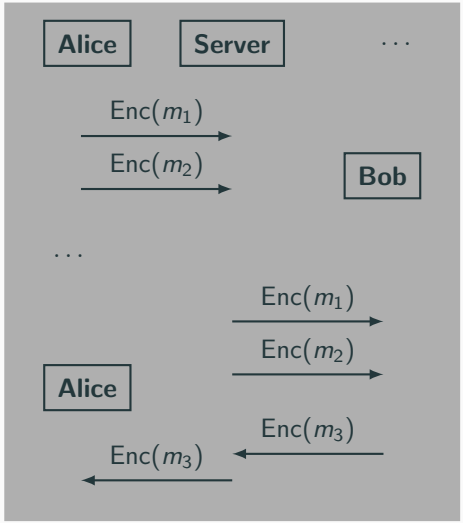


Asynchronous Protocol (3)



Asynchronous messaging protocol:

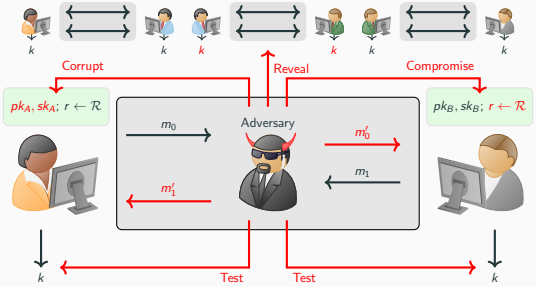
- Asynchronous - Parties not required to be online at the same time
- Server now needed to handle message delivery
- Server delivers messages when parties come online
- Key Establishment needs to be asynchronous!



Signal Threat Model

Security properties inform the discussion about the design of the Signal key exchange protocol. Who is Signal trying to protect against? We consider security against an attacker that:

- is in complete control of the network,
- can **Reveal** derived message keys,
- can **Corrupt** long-term secrets
- can **Compromise** ephemeral secrets



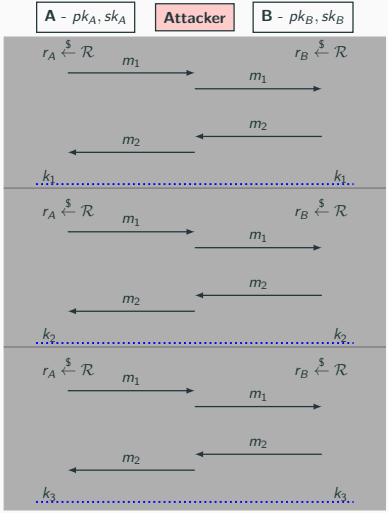
Advanced Security Properties

We want to look at advanced security properties of Authenticated Key Exchange:

- **F**orward **S**ecrecy
- **P**ost **C**ompromise **S**ecurity

Consider an AKE protocol executed in sequence between Alice and Bob

- Alice and Bob have long-term sk_A, sk_B
- Alice and Bob sample ephemeral values r_A, r_B during protocol execution
- Attacker has full network control

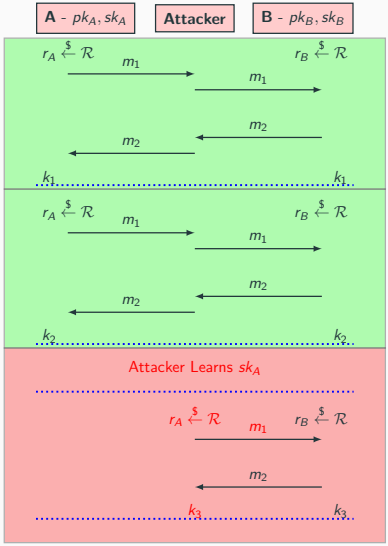


Background: Forward Security

Forward **S**ecurity considers the following scenario:

- Attacker has full control of the network
- Alice and Bob execute AKE together
- Attacker **Corrupts** the long-term key sk_A
- Attacker uses this to impersonate Alice

Obviously, protocol executions with "Alice" after this point are not secure - but what about before Attacker learns sk_A? If key indistinguishability holds, the protocol has forward security

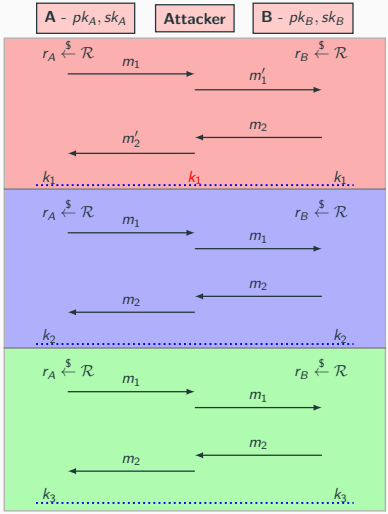


Post-Compromise Security

Post-**C**ompromise **S**ecurity considers the following scenario:

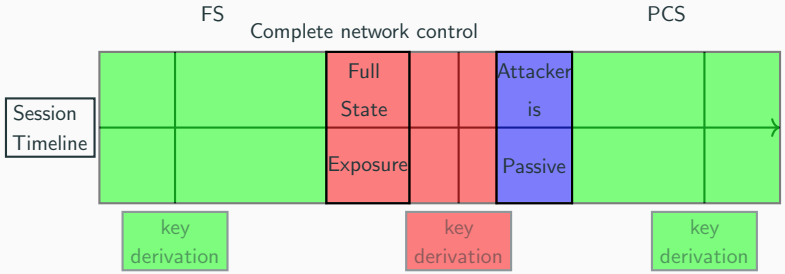
- Attacker has full control of the network
- Attacker **Corrupts** and **Compromises** the full state of Alice, Bob: sk_A, sk_B, r_A, r_B
- Attacker becomes passive
- Alice and Bob execute protocol without modification

If Alice and Bob can recover security as a result of this passivity, preventing the attacker from breaking future executions (even with knowledge of sk_A, sk_B) then the protocol has post-compromise security



Signal aims at FS + PCS

Signal will attempt to achieve both notions simultaneously.



Outline

Secure Messaging and Signal

Signal Overview

The Signal Double Ratchet Protocol

The Signal X3DH Protocol

Extra slides: Signal's Message Encryption and Algorithms

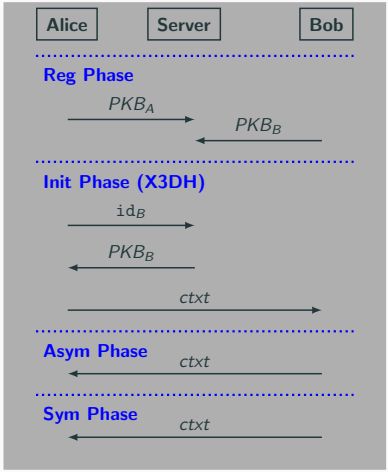
Extra slides: Signal Authentication

Signal On a High-Level

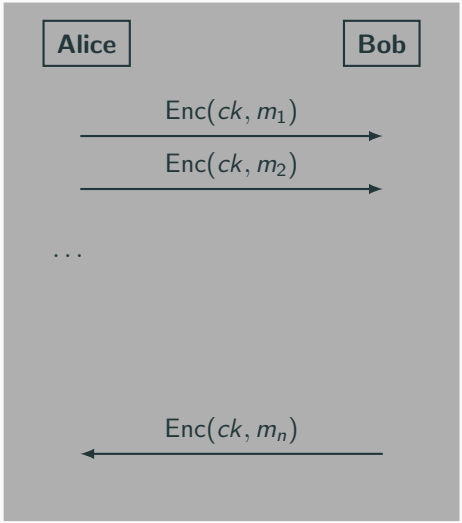
On a high level, the Signal Protocol consists of 4 “distinct” phases.

- A Registration Phase
- An Initialisation Phase
- An Asymmetric Ratchet Phase
- A Symmetric Ratchet Phase

Start at the bottom, and work our way up.

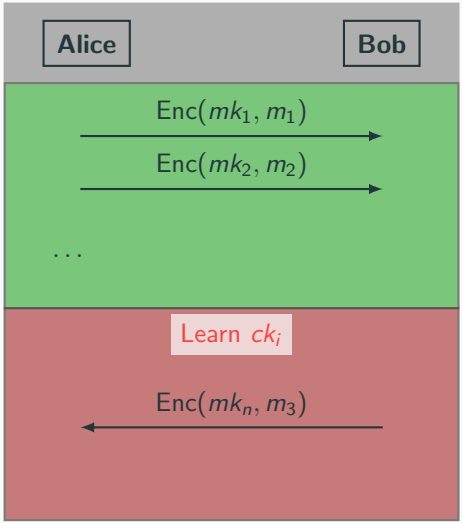


Intuition: Symmetric Ratchet Phase

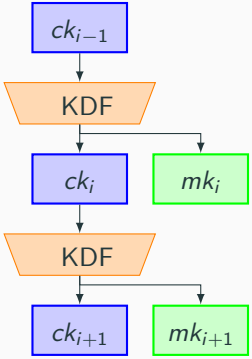


- Let's assume Alice and Bob have already established a shared symmetric key, which we'll call *ck*, or chain key.
- Naive approach:
  Use *ck* to encrypt messages!
- What happens when *ck* is later exposed?
- All security is lost!

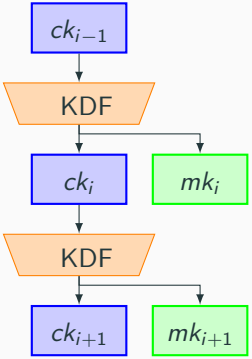
Intuition: Symmetric Ratchet Phase



- Use the KDF to ratchet forward ck_i, and use ck_i only once to derive ck_{i+1}, mk_{i+1}?
- Achieves FS: only messages encrypted after exposure are compromised!

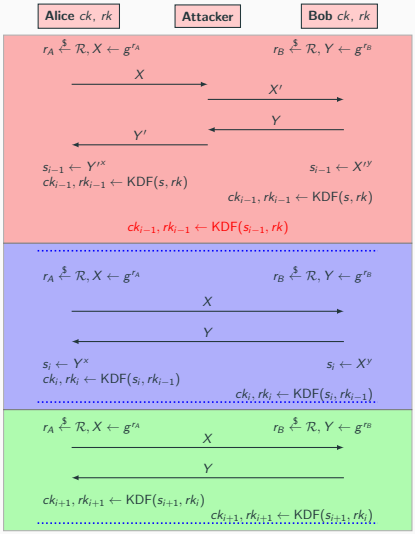


Intuition: Asymmetric Ratchet Phase



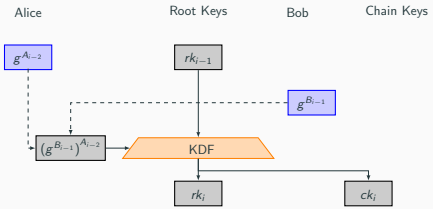
- So, with this symmetric ratchet technique, Signal achieves **FS** with respect to ck (and thus mk)
- But we want to be able to recover security after a compromise i.e **PCS** with respect to ck (and mk)
- How can we do this?

Intuition: Asymmetric Ratchet Phase



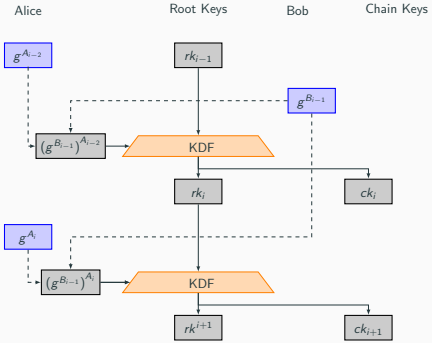
- Let's assume Alice and Bob already have rk, or root key:
  $ck_{i+1}, rk_{i+1} \leftarrow \mathrm{KDF}(rk_i, \ldots)$
- Naive approach: Constantly execute DH, combine rk with DH outputs to derive ck.
- Passive in "blue" $\implies$ rk_i looks random $\implies$ sufficient entropy for "green" KDF
- Problem: Signal is asynchronous - this approach needs both parties to be online.

Solution: Asymmetric Ratchet



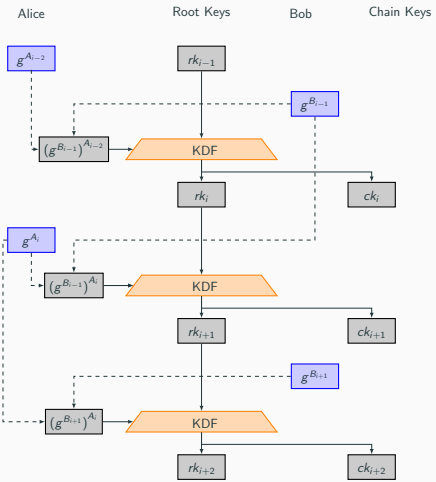
- Assume that Alice and Bob ping-pong messages: A $\rightarrow$ B, then B $\rightarrow$ A, repeat...
- Let's assume Alice has already sent Bob a DH public value $g^{A_{i-2}}$
- Do a half-DH with every message!

Solution: Asynchronous Asymmetric Ratchet



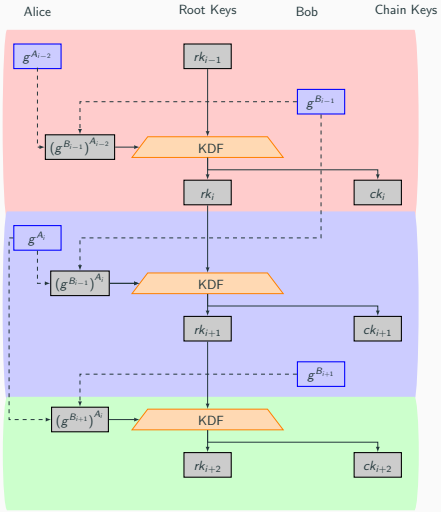
- Assume that Alice and Bob ping-pong messages: A$\rightarrow$ B, then B $\rightarrow$ A, repeat...
- Let's assume Alice has already sent Bob a DH public value $g^{A_{i-2}}$
- Do a half-DH with every message!
- Alice now generates a new DH, and asymmetrically ratchets *rk* forward

Solution: Asynchronous Asymmetric Ratchet



- Assume that Alice and Bob ping-pong messages: A→ B, then B → A, repeat...
- Let's assume Alice has already sent Bob a DH public value $g^{A_{i-2}}$
- Do a half-DH with every message!
- Alice now generates a new DH, and asymmetrically ratchets *rk* forward
- Bob receives Alice's DH g^{A_i} when he comes online
- ... and asymmetrically ratchets again!

Solution: Asynchronous Asymmetric Ratchet



- Assume that Alice and Bob ping-pong messages: A→ B, then B → A, repeat...
- Let's assume Alice has already sent Bob a DH public value $g^{A_{i-2}}$
- Do a half-DH with every message!
- Alice now generates a new DH, and asymmetrically ratchets *rk* forward
- Bob receives Alice's DH g^{A_i} when he comes online
- ... and asymmetrically ratchets again!
- Achieves Post Compromise Security

Outline

Secure Messaging and Signal

Signal Overview

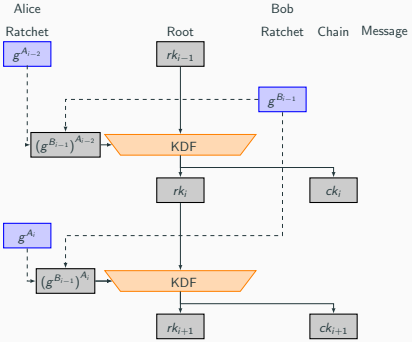
The Signal Double Ratchet Protocol

The Signal X3DH Protocol

Extra slides: Signal's Message Encryption and Algorithms

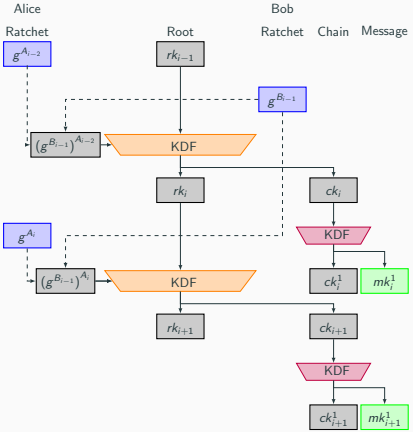
Extra slides: Signal Authentication

Combining Asymmetric and Symmetric: The Double Ratchet Protocol



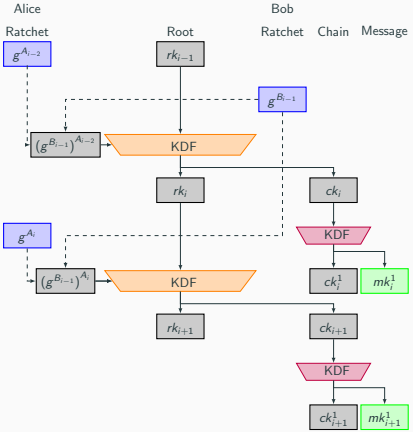
- Combining the Asymmetric Ratchet and Symmetric Ratchet is straightforward
- Asymmetric Ratchet computes new rk, ck

Combining Asymmetric and Symmetric: The Double Ratchet Protocol



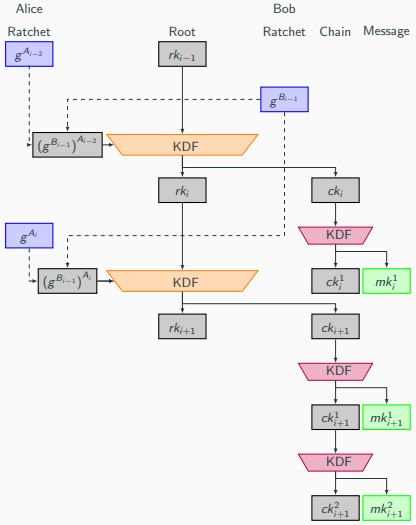
- Combining the Asymmetric Ratchet and Symmetric Ratchet is straightforward
- Asymmetric Ratchet computes new *rk*, *ck*
- Symmetric Ratchet computes new *ck*, *mk*
- Use *mk* to encrypt messages!

Combining Asymmetric and Symmetric: The Double Ratchet Protocol



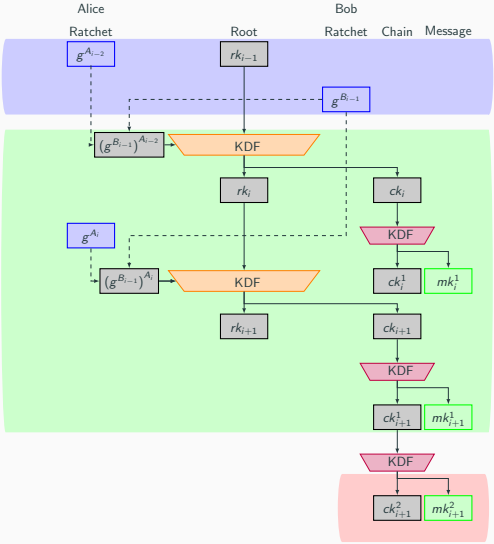
- What if Alice and Bob don't ping-pong?
- Alice wants to send another message, but Bob won't come online and send new DH
- Generates g^{A_i}, asymm. ratchets rk_i, ck_i
- Computes ck_{i+1}^1, mk_{i+1}^1 to send Bob $\text{Enc}(mk_{i+1}^1, m_{i+1}^1)$

Combining Asymmetric and Symmetric: The Double Ratchet Protocol



- What if Alice and Bob don't ping-pong?
- Alice wants to send another message, but Bob won't come online and send new DH
- Generates g^{A_i}, asymm. ratchets rk_i, ck_i
- Computes ck_{i+1}^1, mk_{i+1}^1 to send Bob $\text{Enc}(mk_{i+1}^1, m_{i+1}^1)$
- Alice symm. ratchets ck_{i+1}^2, mk_{i+1}^2 to send Bob $\text{Enc}(mk_{i+1}^2, m_{i+1}^2)$
- and so on...
- High level: Asymmetric ratchet happens on ping-pong, Symmetric ratchet happens on successive messages

Combining Asymmetric and Symmetric: The Double Ratchet Protocol



- What if Alice and Bob don't ping-pong?
- Alice wants to send another message, but Bob won't come online and send new DH
- Generates g^{A_i}, asymm. ratchets rk_i, ck_i
- Computes ck_{i+1}^1, mk_{i+1}^1 to send Bob $\text{Enc}(mk_{i+1}^1, m_{i+1}^1)$
- Alice symm. ratchets ck_{i+1}^2, mk_{i+1}^2 to send Bob $\text{Enc}(mk_{i+1}^2, m_{i+1}^2)$
- and so on...
- High level: Asymmetric ratchet happens on ping-pong, Symmetric ratchet happens on successive messages

Outline

Secure Messaging and Signal

Signal Overview

The Signal Double Ratchet Protocol

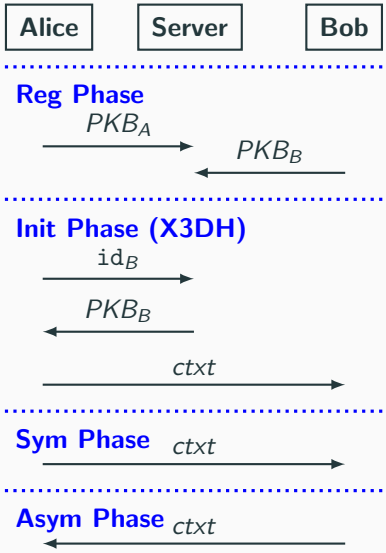
The Signal X3DH Protocol

Extra slides: Signal's Message Encryption and Algorithms

Extra slides: Signal Authentication

Signal Overview

- At this point, it's worth revisiting what we've looked at
- We assume Alice and Bob share some secret state *rk*
- We examined how Signal uses that *rk* to establish per-message secrets via the Double Ratchet Protocol
- How to get initial *rk*?
- Registration Phase: Parties upload public keys to Server
- Initialisation Phase: Parties use public keys produce initial *rk*



Signal Registration Phase

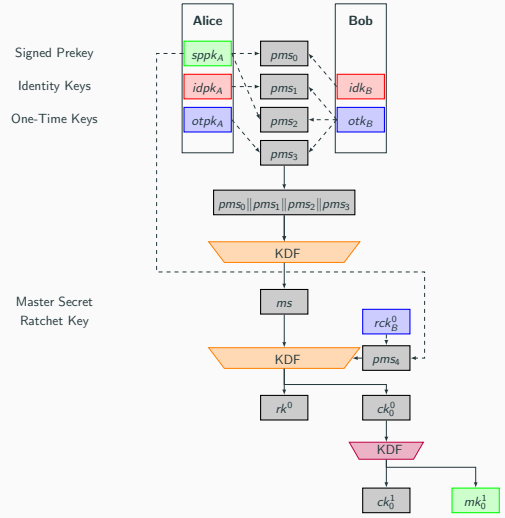
Alice wants people to be able to message her. To do so, she generates a PreKeyBundle:

- $\texttt{id}_A$ - long-term identifier (Phone number)
- $(idk_A, idpk_A = g^{idk_A}) \xleftarrow{\$}$ DHGen - long-term public-key pair (Identity key)
- $(spk_A, sppk_A = g^{spk_A}) \xleftarrow{\$}$ DHGen - medium-term public-key pair (Signed Prekey)
- $(otk_A, otpk_A = g^{otk_A}) \xleftarrow{\$}$ DHGen - ephemeral public-key pair (One-time Key)
- $\sigma_A \xleftarrow{\$} \text{SIG.Sign}_{idk_A}(sppk_A)$ - signature
- $\underline{\text{PKB}}_A = \{\texttt{id}_A,\ idpk_A, sppk_A, otpk_A, \sigma_A\}$

Alice uploads the PreKeyBundle to the Signal Server.
Note: No authentication mechanism, no PKI.

Signal Initialisation Phase (X3DH KEX)

- Bob wants to communicate with Alice.
- Bob gets Alice's PKB_A from Server
- Bob generates an ephemeral share $(otk_B, g^{otk_B} = otpk_B) \xleftarrow{\$}$ DHGen
- Bob combines PKB_A with its idk_B and otk_B in what's called the e**X**tended triple (**3**) **D**iffie-**H**ellman key exchange to produce a symmetric **m**aster **s**ecret ms.
- Bob can then initiate the first asymmetric/symmetric ratchet and send the first message, along with $otpk_B$.
- Note: Only Bob acting ($\rightarrow$ asynchronous). Alice can derive ms later, knowing Bob's $idpk_B$ and receiving $otpk_B$.

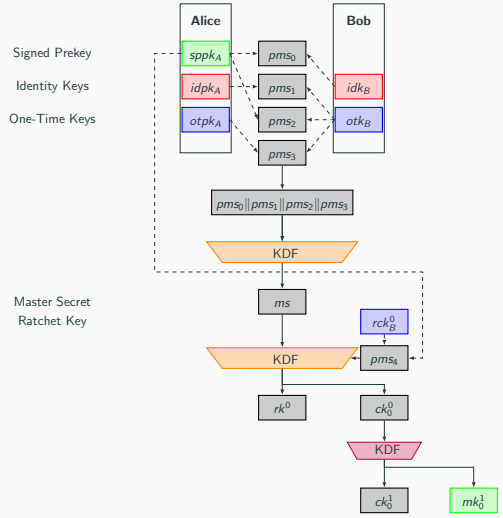


Signal Initialisation Phase (X3DH KEX)

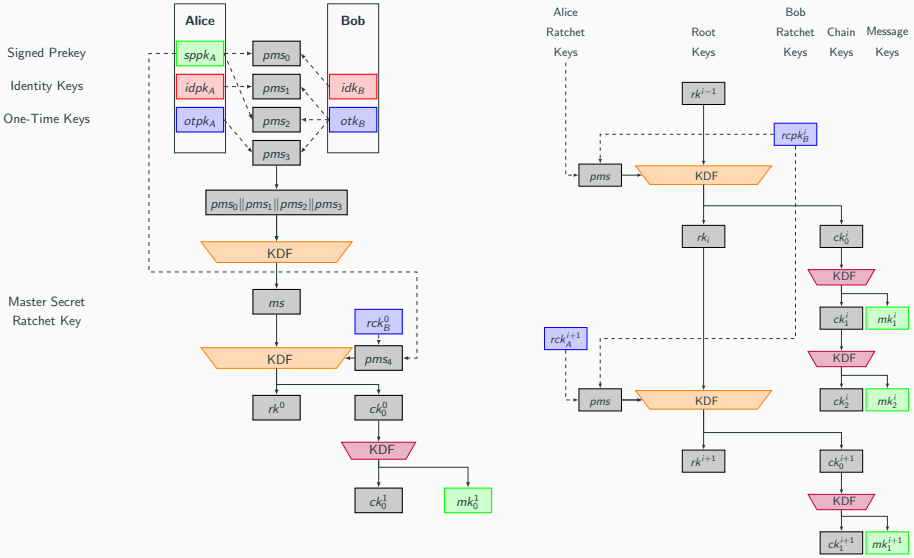
Why this combination of DH values to produce the master secret?

- otk_A-otk_B provides forward security
- $sppk_A$-otk_B provides (delayed) forward security if no otk_A is left in Alice's PKB
- $sppk_A$-idk_B provides Bob authentication
- $idpk_A$-otk_B provides Alice authentication
- Lack of $idpk_A$-idk_B provides a notion of deniability: if otk, spk_A leak later, then anyone could have generated this

No binding of identities to public keys – Unknown Key Share attacks possible.

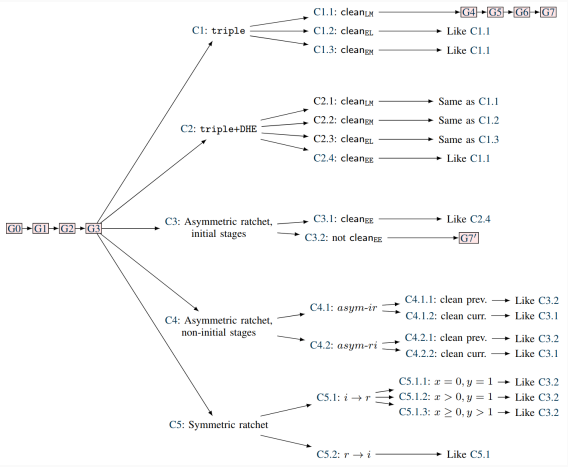


The Signal Protocol



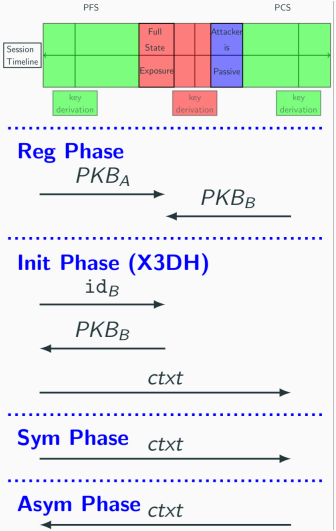
Security Analysis of the Signal Protocol

- CCDGS17. "A Formal Security Analysis of the Signal Messaging Protocol" – First formal analysis of the Signal Protocol in computational model
- KBB2017. "Automated verification for secure messaging protocols and their implementations"
- RMS2018. "More is less: on the end-to-end security of group chats in Signal, WhatsApp, and Threema."
- ACY2019. "The Double Ratchet: Security notions, proofs, and modularization for the Signal Protocol."
- Still a very active area!



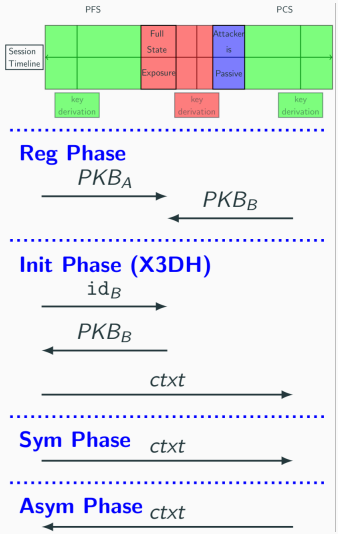
Summary time!

- Two-party asynchronous E2EE messaging
- Discussed the security properties that Signal hopes to achieve
- Specifically looked at Forward Security and Post Compromise Security
- Took an in-depth look at the Signal Double Ratchet and X3DH Key Exchange protocols



What We Didn't Talk About

- The Multi-Device Setting - How does Signal share keys securely?
- Group Messaging - How does 2 Party become N-Party?
- Signal Authentication - Safety Numbers and Out-of-Band Verification
- Private Contact Discovery and Private Groups
- Formalising Ratcheted Key Exchange and Secure Messaging



Outline

Secure Messaging and Signal

Signal Overview

The Signal Double Ratchet Protocol

The Signal X3DH Protocol

Extra slides: Signal's Message Encryption and Algorithms

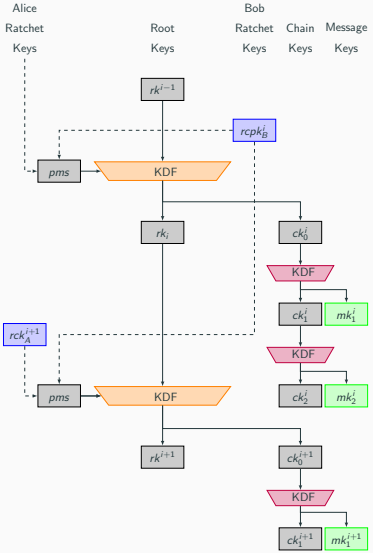
Extra slides: Signal Authentication

Wait, you're just talking about keys! You didn't talk about MESSAGING!

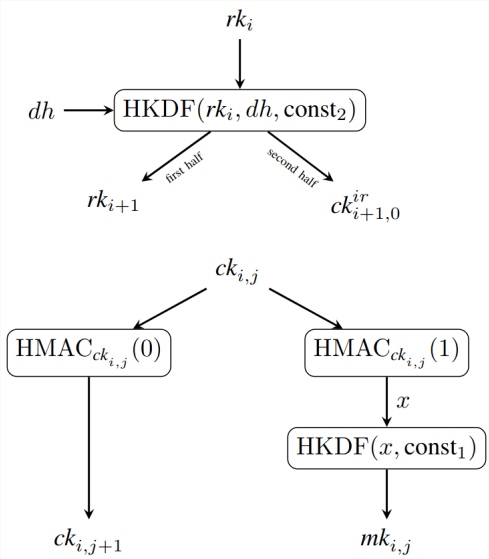
- How does Signal actually encrypt messages?
- Signal uses AEAD encryption: recall that the associated data (AD) is authenticated but not encrypted.
- Message key mk we've already discussed, plaintext is the Signal message - What is AD?
- $AD = rcpk_A^i \| ipk_A \| ipk_B \| PN \| ctr$
- What does this mean?

$AD = rck_A^i \| ipk_A \| ipk_B \| PN \| ctr$: Cool features

- $rcpk_A^i$ is the most recent ratchet (public) key generated by the encrypting party
- ipk_A, ipk_B the long-term public keys (or public key identifier) of Alice and Bob
- PN is the number of messages in the last chain sent by the encrypting party (allows decrypter to delete old ck values)
- ctr is the current number of messages sent by the encrypting party in the current chain (allows out-of-order messages without trial decryption)



Signal In Practice



- **Key Derivation**: Root KDF uses HKDF-SHA256, Chain KDF uses HMAC-SHA256, HKDF-SHA256
- **Hash Functions**: SHA256
- **AEAD Encryption** by way of **Encrypt-then-MAC** approach: AES256-CBC with PKCS#7 padding, HMAC-SHA256 for MAC
- **Signature Scheme**: Ed25519
- **Diffie-Hellman**: Elliptic curve with X25519 or X448

Outline

Secure Messaging and Signal

Signal Overview

The Signal Double Ratchet Protocol

The Signal X3DH Protocol

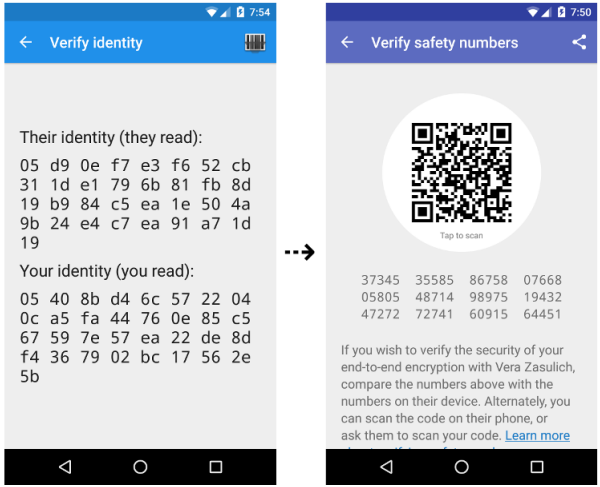
Extra slides: Signal's Message Encryption and Algorithms

Extra slides: Signal Authentication

Signal Authentication

- Recall the structure of the Prekey Bundle:
  $\underline{\text{PKB}}_B = \{\texttt{id}_B,\ idpk_B, sppk_B, otpk_B, \sigma_B\}$.
- The Signal Server can simply generate their own $idpk'_B$, compute their own PreKeyBundle, and imitate Bob to Alice.
- How does Signal prevent this?

Capturing Users in Key Exchange Protocols



Safety numbers!

Contains only static, publicly computable values.

$$\texttt{local\_fprint} = H_i{}^1(0\|\texttt{fvers}\|idpk_A\|\texttt{id}_A, idpk_A)$$

$$\texttt{remote\_fprint} = H_i(0\|\texttt{fvers}\|idpk_B\|\texttt{id}_B, idpk_B)$$

$$safetynumber = \texttt{local\_fprint}\|\texttt{remote\_fprint}$$

However, you'll notice that the safety on the previous page is 60 decimal digits, whereas this should be over 300. What gives?

[1] $H_i(x, y)$ is an iterated hash, where $H_0 = H(x)$, and $H_i = H(H_{i-1}\|y)$. In the repository we examined, the Signal AP uses SHA2 with 512-bits of output as the underlying hash function, with 5200 iterations.

Safety number truncation