# Privacy Enhancing Technologies (BSc)

## Lecture VIII: Messaging II

Univ.-Prof. Dr. Dominique Schröder

January 13, 2025

# Overview

| KW | Type | Lectures |
|---|---|---|
| 42 | **Lecture** | Internet and Privacy in the Web |
| 43 | **Lecture** | Fingerprinting & Censorship |
| 44 | **Exercise** | Introduction to Rust & Challenge Start |
| 45 | **Lecture** | Cryptographic Preliminaries |
| 46 | **Lecture** | Public Key Cryptography and Security Reductions |
| 47 | **Exercise** | Proofs and Theoretical Deepening |
| 48 | **Lecture** | TLS |
| 49 | **Exam** | **Midterm Exam** |
| 50 | **Exercise** | Challenge Hands-On |
| 51 | **Lecture** | Messaging I |
| **3** | **Lecture** | **Messaging II (Forward Security, Post Compromise Security)** |
| 4 | **Lecture** | Onion |
| 5 | **Lecture** | Q & A |
| 6 | **Exam** | **Final Exam** |
| 9 | **Exam** | **Reexam** (Both for Midterm and Final) |



TU WIEN
PRIVACY ENHANCING TECHNOLOGIES

# This Lecture's Agenda

- Recap: Security Definition and Trivial Winning
- Forward Security
- Post Compromise Security
- The Signal Protocol

# Security Model - Synchronized

## Unidirectional Messaging

**Game.IND$^{b}_{\mathsf{URKE},\mathcal{A}}(\lambda)$**

1 : $s \leftarrow 0; r \leftarrow 0; sync \leftarrow 1$

2 : $C[\cdot] \leftarrow \perp$

3 : $(st_S, st_R) \leftarrow init()$

4 : $b' \leftarrow \mathcal{A}^{\mathcal{O}}(1^{\lambda})$

5 : **Stop with** $b'$

**Oracle.Snd$^{b}()$**

1 : $(st_S, K_0, c) \leftarrow \mathsf{snd}(st_S)$

2 : $K_1 \leftarrow\!\$\ \mathcal{K}$

3 : $C[s] \leftarrow c$

4 : $s \leftarrow s + 1$

5 : **return** $(K_b, c)$

**Oracle.Rcv$(c)$**

1 : $(st_R, K_0) \leftarrow \mathsf{rcv}(st_R, c)$

2 : **if** $c \neq C[r] \wedge sync = 1$ :

3 : $sync \leftarrow 0$

4 : $r \leftarrow r + 1$

5 : **if** $sync = 1$ :

6 : **return** $\perp$

7 : **return** $K_0$

**Oracle.ExposeS()**

1 : **return** $st_S$

**Oracle.ExposeR()**

1 : **return** $st_R$

# Security Model - Trivial Winning Conditions

## Unidirectional Messaging

Before we can define what security means, we have to identify and exclude certain trivial winning conditions. Under trivial conditions, it is easy to win the game, even for efficient adversaries. However, these attacks are meaningless in practice.

By fine-tuning the trivial winning conditions, we can fine-tune the security guarantees of our protocol. Possible examples of trivial winning conditions would be:

- The adversary exposes the receiver and queries the send oracle afterward.
- The adversary queries the send oracle and exposes the receiver before receiving the ciphertext.

### Excluded Out-Of-Sync Winning

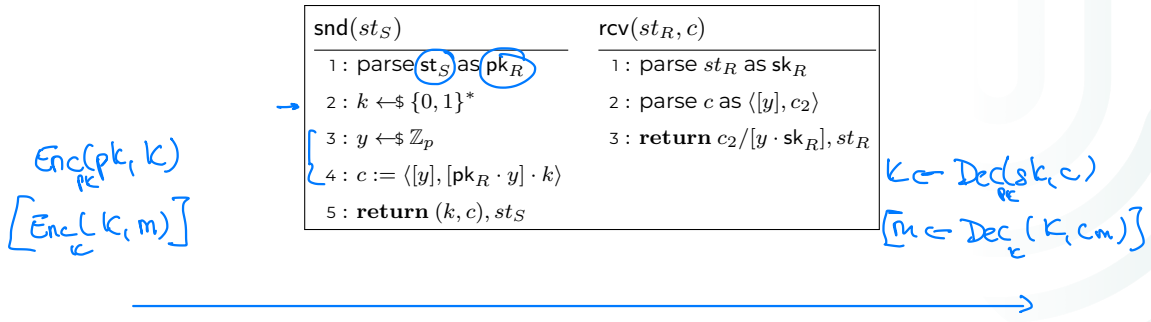
It is not a trivial winning to learn the key $K_0$ through the receive oracle when the system is out of sync since, in this case, we want incompatible states to not lead to meaningful keys.



TU WIEN
PRIVACY ENHANCING TECHNOLOGIES

# First Idea: El-Gamal

## A Secure Construction? (1)

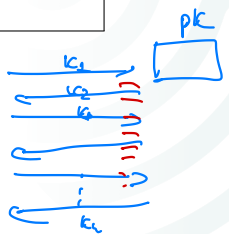


# First Idea: El-Gamal

## A Secure Construction? (2)

| $\mathsf{Game.IND}^{\mathsf{b}}_{\mathsf{URKE},\mathcal{A}}(\lambda)$ | $\mathsf{Oracle.Snd}^{\mathsf{b}}()$ | $\mathsf{Oracle.Rcv}(c)$ |
|---|---|---|
| 1 : $s \leftarrow 0; r \leftarrow 0; sync \leftarrow 1$ | 1 : $(st_S, K_0, c) \leftarrow \mathsf{snd}(st_S)$ | 1 : $(st_R, K_0) \leftarrow \mathsf{rcv}(st_R, c)$ |
| 2 : $C[\cdot] \leftarrow \bot$ | 2 : $K_1 \leftarrow\$ \mathcal{K}$ | 2 : **if** $c \neq C[r] \wedge sync = 1$ : |
| 3 : $(st_S, st_R) \leftarrow init()$ | 3 : $C[s] \leftarrow c$ | 3 : $sync \leftarrow 0; \quad r \leftarrow r + 1$ |
| 4 : $b' \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda)$ | 4 : $s \leftarrow s + 1$ | 4 : **if** $sync = 1$ : **return** $\bot$ |
| 5 : **Stop with** $b'$ | 5 : **return** $(K_b, c)$ | 5 : **return** $K_0$ |

| $\mathsf{snd}(st_S)$ | $\mathsf{rcv}(st_R, c)$ |
|---|---|
| 1 : parse $\mathsf{st}_S$ as $\mathsf{pk}_R$ | 1 : parse $st_R$ as $\mathsf{sk}_R$ |
| 2 : $k \leftarrow\$ \{0,1\}^*$ | 2 : parse $c$ as $\langle [y], c_2 \rangle$ |
| 3 : $y \leftarrow\$ \mathbb{Z}_p$ | 3 : **return** $c_2 / [y \cdot \mathsf{sk}_R], st_R$ |
| 4 : $c := \langle [y], [\mathsf{pk}_R \cdot y] \cdot k \rangle$ | |
| 5 : **return** $(k, c), st_S$ | |

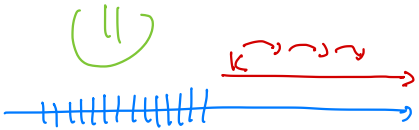


# First Idea: El-Gamal

## A Secure Construction? (3)

**All past exchanged keys are lost when a state is revealed!**

| $\mathsf{snd}(st_S)$ | $\mathsf{rcv}(st_R, c)$ |
|---|---|
| 1 : parse $\mathsf{st}_S$ as $\mathsf{pk}_R$ | 1 : parse $st_R$ as $\mathsf{sk}_R$ |
| 2 : $k \leftarrow\$ \{0,1\}^*$ | 2 : parse $c$ as $\langle [y], c_2 \rangle$ |
| 3 : $y \leftarrow\$ \mathbb{Z}_p$ | 3 : **return** $c_2 / [y \cdot \mathsf{sk}_R], st_R$ |
| 4 : $c := \langle [y], [\mathsf{pk}_R \cdot y] \cdot k \rangle$ | |
| 5 : **return** $(k, c), st_S$ | |

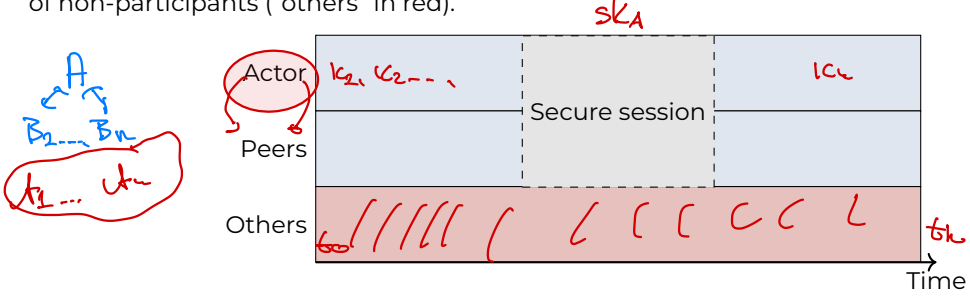



# Forward Security

# Forward Security

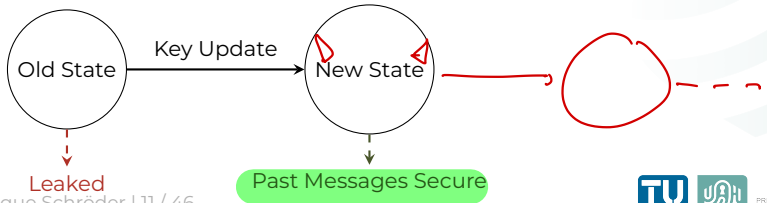
## Classical adversary model

If Alice is communicating with Bob, classical models allow the adversary to compromise the long-term keys of everyone except Alice and Bob. This ability to compromise the long-term key of non-participants (“others” in red).



# What is Forward Security?

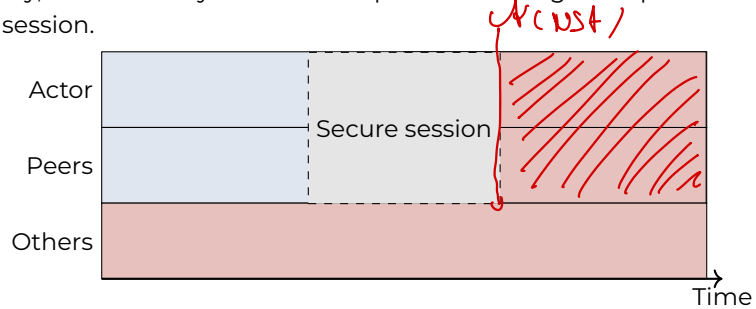
- **Motivation:**
  - Leakage of the current secret key can compromise all previously transmitted messages.
  - Adversaries can decrypt past communications, violating confidentiality.
- **Key Idea:**
  - Forward security ensures previously transmitted messages remain secure even if the current state is compromised.
  - Achieved by **frequently updating cryptographic states**, ensuring new states are independent of previous ones.



# Forward Security

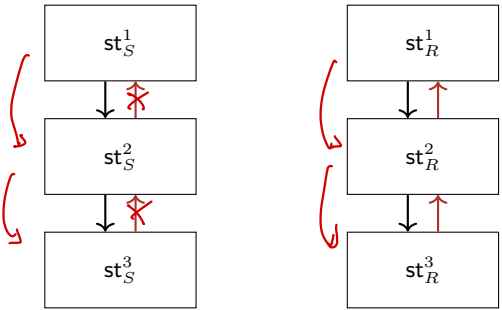
## Forward Secure Model

In forward security, the adversary can also compromise all long-term private keys after the end of the attacked session.



# Updating Our Running Example

## Forward Secure States




- Each time a party sends (or receives) a message, it updates its internal state.
- From state $i$ it is possible to compute state $i+1$, but from state $i$ it is infeasible to compute state $i-1$.
- Decrypting message $i$ works only with state $i$.

# Linear State El-Gamal

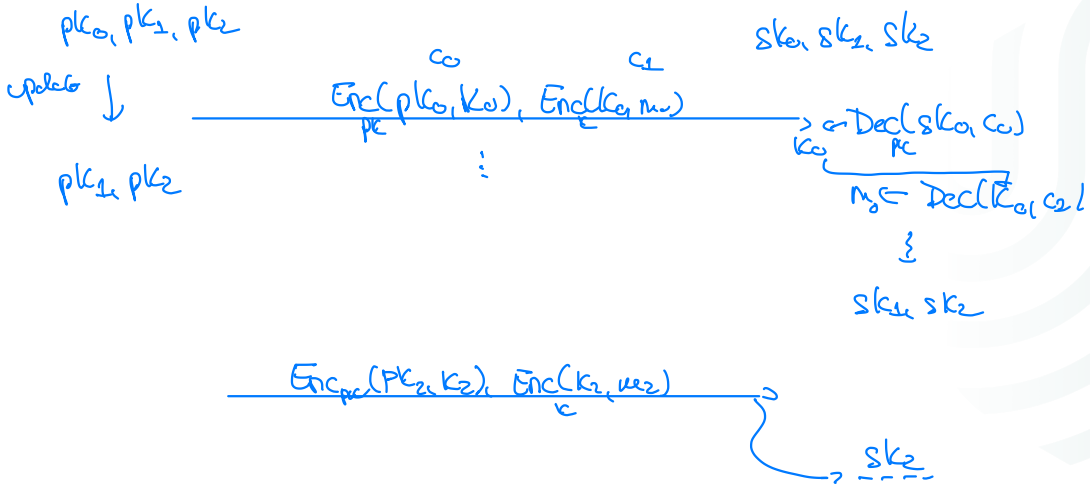
## A Secure Construction? (1)

| $\mathsf{snd}(st_S)$ | $\mathsf{rcv}(st_R, c)$ |
|---|---|
| 1 : parse $\mathsf{st}_S$ as $\{\mathsf{pk}_{R_i}, \ldots, \mathsf{pk}_{R_n}\}$ | 1 : parse $st_R$ as $\{\mathsf{sk}_{R_i}, \ldots, \mathsf{sk}_{R_n}\}$ |
| 2 : $k \leftarrow\$ \{0,1\}^*$ | 2 : parse $c$ as $\langle [y], c_2 \rangle$ |
| 3 : $y \leftarrow\$ \mathbb{Z}_p$ | 3 : $\mathsf{st}_R \leftarrow \{\mathsf{sk}_{R_{i+1}}, \ldots, \mathsf{sk}_{R_n}\}$ |
| 4 : $c := \langle [y], [\mathsf{pk}_{R_i} \cdot y] \cdot k \rangle$ | 4 : **return** $c_2 / [y \cdot \mathsf{sk}_R], st_R$ |
| 5 : $\mathsf{st}_S \leftarrow \{\mathsf{pk}_{R_{i+1}}, \ldots, \mathsf{pk}_{R_n}\}$ | |
| 6 : **return** $(k, c), st_S$ | |

- **Pro:** When the receiver state $i$ is leaked, all keys *before* the leak are secure.
- **Con:** The state is linear in the number of exchanged messages.
- **Remaining:** No Recovery *after* a key leakage.

# Linear State El-Gamal

## A Secure Construction? (2)



# Post-Compromise Security

# Post-Compromise Security

- If Bob's current state (the decapsulation key) gets leaked, also all future keys would be known to an attacker
- We want new keys to be secure again after Bob was compromised
- This is called **Post-Compromise Security**
- Sadly, in the unidirectional setting, Post-Compromise Security can't be achieved because of correctness:
- All Bob does is receive; that's why his receive algorithm is (and has to be) purely deterministic.
  It follows that an attacker who knows Bob's current state can also derive all future states and, therefore, will be able to execute rcv with Bob's state and receive all future keys.

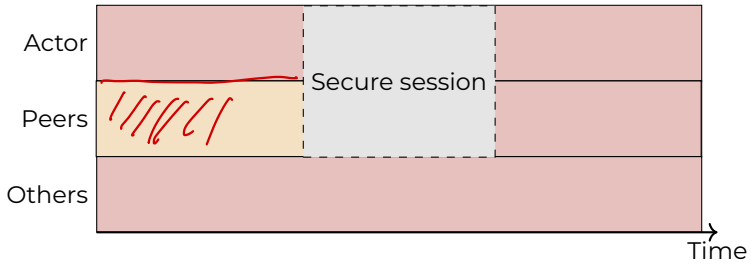


TU WIEN PRIVACY ENHANCING TECHNOLOGIES

# Post-Compromise Security

## Possible Models (1)

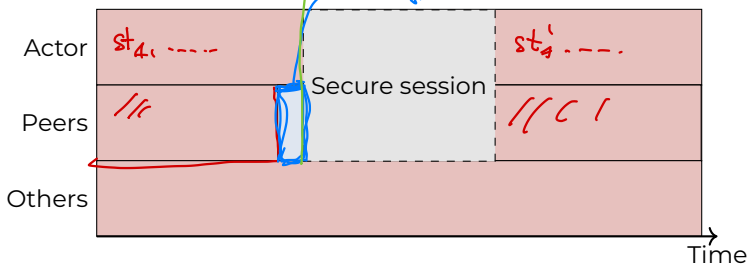

We denote with the orange box (the left side of the “peers” row) a limited form of compromise, which allows the adversary some extra power without revealing the long-term key.



# Post-Compromise Security

## Possible Models (2)

Another form of PCS can be achieved if the long-term key is compromised but there is at least one uncompromised session afterwards.




One can view our stronger form of PCS as a form of “bootstrapping”, whereby secrecy of one session key K is used to ensure the secrecy of its successors.

# Post-Compromise Security

- Post-Compromise Security is, however, achievable in the bi-directional setting where Bob is also allowed to send messages to Alice
- In this setting, he can do random updates of his state and notify Alice about the changes
- This way, an adversary with knowledge of one of Bob's previous states is not able to perform the same updates because he doesn't have access to Bob's randomness

creating fresh and independent keys



TU WIEN
PRIVACY ENHANCING TECHNOLOGIES

# Updating Our Running Example

## Post-Compromise Secure El-Gamal

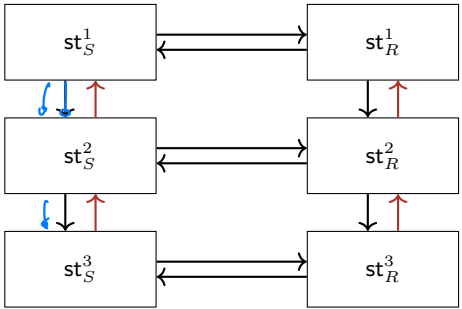
- To achieve post-compromise security in our El-Gamal example, we can make use of iterated key exchanges:
  - In each interaction, the respective sender sends a fresh ephemeral public key, which is included in the shared key.
  - A single honest round “locks an adversary out”.



TU WIEN
PRIVACY ENHANCING TECHNOLOGIES

# Updating Our Running Example

## Forward Secure States




- Each time a party sends (or receives) a message, it updates its internal state by an ephemeral key exchange.
- From state $i$ it is only possible to compute state $i+1$ via interaction, but this update is probabilistic.
- Decrypting message $i$ works only with state $i$.

We defer an example construction after the next section.

# Signal Protocol

- There's still one property in our security definition that we desire for our messaging protocol:
- We want the sender and receiver state to become incompatible as soon as the adversary alters a message.
- This way an an active attacker won't stay unnoticed.



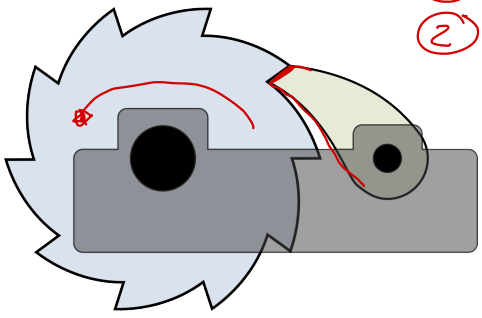
TU WIEN
PRIVACY ENHANCING TECHNOLOGIES

# Double Ratchet

# Single Ratchet Mechanism

## What is a Single Ratchet? (1)

- A **Single Ratchet** is a cryptographic mechanism used to update keys securely over time.
- Ensures:
  - **Forward Security:** Past messages remain secure even if current keys are compromised.
  - Frequent key updates using a deterministic process.



# Single Ratchet Mechanism

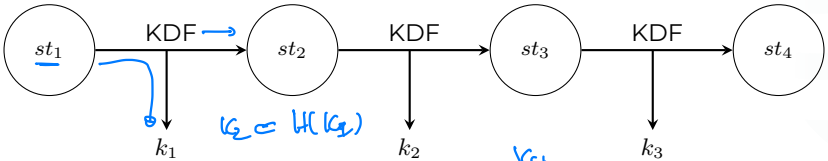
## What is a Single Ratchet? (2)

**Key Idea:**

- Derive a new key for each message using a Key Derivation Function (KDF).
- Each state is derived from the previous state:

$$st_{i+1} = \mathsf{KDF}(st_i)$$

- The new key depends only on the current key, ensuring irreversibility.



# What is the Double Ratchet?

- **Motivation:**
  - Enhance forward security by integrating message-level state updates.
  - Provide **Post-Compromise Security** by combining **Diffie-Hellman ratchets** with **key derivation functions (KDFs)**.
- **Key Idea:**
  - Each message exchange triggers a new key derivation.
  - Future keys depend on both parties' current states, ensuring strong security properties.

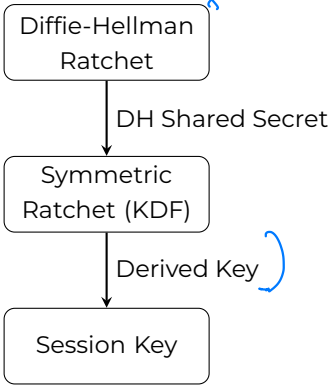


TU WIEN
PRIVACY ENHANCING TECHNOLOGIES

# Double Ratchet Architecture

## Overview of Components

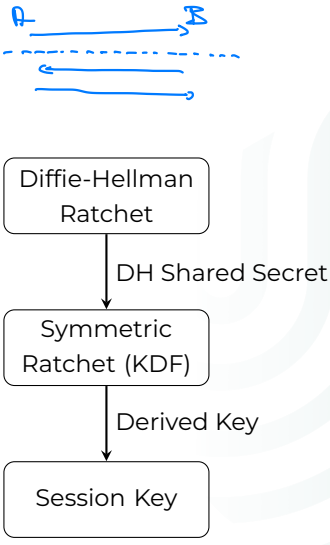
The Double Ratchet combines a Diffie-Hellman Ratchet for major updates and a Symmetric Ratchet for message-level updates.



# Double Ratchet Algorithm

## How it Works

- **Initialization:**
  - Generate initial Diffie-Hellman keys and shared secrets.
  - Set up KDF chains for sending and receiving.
- **Sending a Message:**
  - Update Diffie-Hellman keys if required.
  - Use the sending KDF chain to derive a message key.
- **Receiving a Message:**
  - Use the receiving KDF chain to derive the corresponding message key.
  - Update the receiving chain after decryption.



# Double Ratchet in Action


## DH-Ratchet of Alice

- Initially, Alice knows the public key of Bob $\mathsf{pk}_B$ and her own secret key $\mathsf{sk}_A \rightarrow \mathsf{DH}(\mathsf{sk}_A, \mathsf{pk}_B)$.
- Alice sends an ephemeral key to Bob leading into the DH state:

$$\mathsf{DH}(\mathsf{sk}_A, \mathsf{pk}_B), \mathsf{DH}(\mathsf{esk}_A, \mathsf{pk}_B)$$

First MSG

- Once Bob answers Alice, he also sends an ephemeral key to Alice:

$$\mathsf{DH}(\mathsf{sk}_A, \mathsf{pk}_B), \mathsf{DH}(\mathsf{esk}_A, \mathsf{pk}_B), \mathsf{DH}(\mathsf{sk}_A, \mathsf{epk}_B), \mathsf{DH}(\mathsf{esk}_A, \mathsf{epk}_B)$$

epk

- Whenever they want to turn the ratchet again, they exchange new ephemeral keys and update the existing ratchet state by the value

$$\mathsf{DH}(\mathsf{esk}_A^i, \mathsf{epk}_B^i)$$

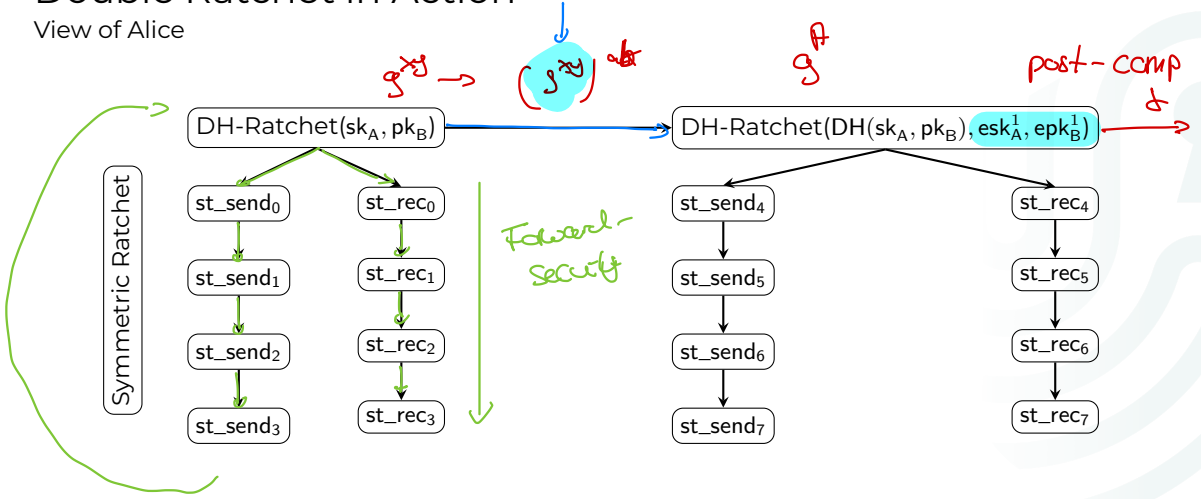
This DH ratched enables PCS, since once a fresh DH pair is included in the state, all successor states are secure again.



TU WIEN
PRIVACY ENHANCING TECHNOLOGIES

# Double Ratchet in Action

View of Alice



# Double Ratchet in Action

## Symmetric Ratchet of Alice

- In each state of the DH ratchet, Alice and Bob start to send and receive messages.
- They do this by using two symmetric ratchets that guarantee forward security:
- They initialize the ratchet state using the state of the DH ratchet (with separated domains).
- Sending:
  - $(\mathsf{ek}_{i+1}, \mathsf{st\_send}_{i+1}) \leftarrow \mathsf{H}(\mathsf{st\_send}_i)$.
  - Use the key $\mathsf{ek}_{i+1}$ for encryption.
  - Use the state $\mathsf{st\_send}_{i+1}$ as new sender state.
- Receiving:
  - $(\mathsf{dk}_{i+1}, \mathsf{st\_rec}_{i+1}) \leftarrow \mathsf{H}(\mathsf{st\_rec}_i)$.
  - Use the key $\mathsf{dk}_{i+1}$ for decryption.
  - Use the state $\mathsf{st\_rec}_{i+1}$ as new receiver state.
- This allows Alice and Bob to simultaneously send and receive messages. When messages are replaced, the states diverge.

# Security Benefits of Double Ratchet

- **Forward Security:** Each symmetric key update ensures that a compromised key does not affect previously exchanged messages.
- **Post-Compromise Security:** Each invocation of the DH ratched secures keys after a key compromise.
- **Replay Prevention:** Each message has a unique key, preventing re-use or tampering.

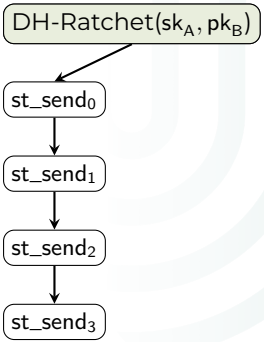


TU WIEN
PRIVACY ENHANCING TECHNOLOGIES

# Double Ratchet in Action

## View of Alice Before Bob Answers

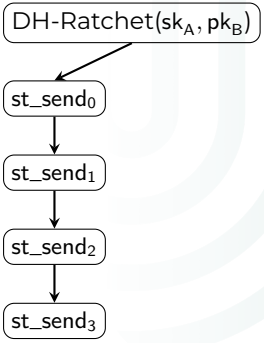
- Alice has a secret key and knows the public key of Bob.
- She initializes the DH Ratchet with these values:
  $st_A = [\mathsf{sk}_A, \mathsf{pk}_B], \mathsf{sk}_A$
- Alice can start sending messages to Bob by turning the symmetric ratchet:
  $ek_0 = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], A, B)$
- …
- $ek_3 = \mathsf{KDF}(\mathsf{KDF}(\mathsf{KDF}(\mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], A, B))))$



# Double Ratchet in Action

## View of Alice Before Bob Answers

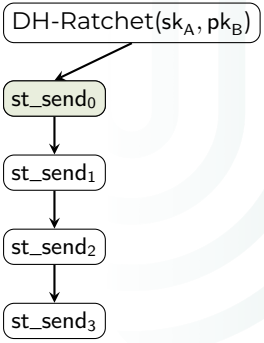
- Alice has a secret key and knows the public key of Bob.
- She initializes the DH Ratchet with these values:
  $st_A = ([sk_A, pk_B], sk_A$
- Alice can start sending messages to Bob by turning the symmetric ratchet:
  $ek_0 = KDF([sk_A, pk_B], A, B)$
- …
- $ek_3 = KDF(KDF(KDF(KDF([sk_A, pk_B], A, B))))$



# Double Ratchet in Action

## View of Alice Before Bob Answers

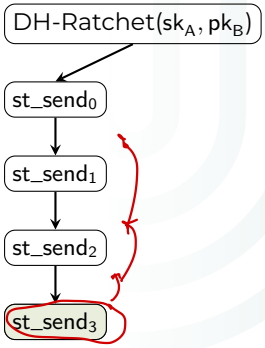
- Alice has a secret key and knows the public key of Bob.
- She initializes the DH Ratchet with these values:
  $st_A = [\mathsf{sk}_A, \mathsf{pk}_B], \mathsf{sk}_A$
- Alice can start sending messages to Bob by turning the symmetric ratchet:
  $ek_0 = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], A, B)$
- …
- $ek_3 = \mathsf{KDF}(\mathsf{KDF}(\mathsf{KDF}(\mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], A, B))))$



# Double Ratchet in Action

## View of Alice Before Bob Answers

- Alice has a secret key and knows the public key of Bob.
- She initializes the DH Ratchet with these values:
  $st_A = [\mathsf{sk}_A, \mathsf{pk}_B], \mathsf{sk}_A$
- Alice can start sending messages to Bob by turning the symmetric ratchet:
  $ek_0 = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], A, B)$ round 0
- …
- $ek_3 = \mathsf{KDF}(\mathsf{KDF}(\mathsf{KDF}(\mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], A, B))))$ round 3, round 2



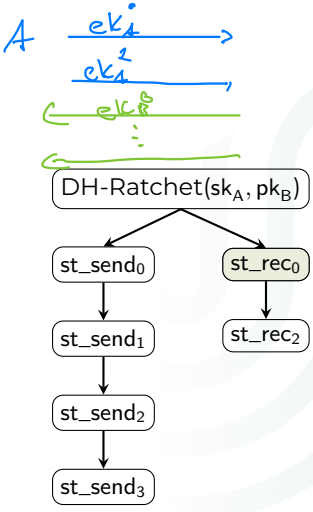
# Double Ratchet in Action

## View of Alice When Bob Answers

- When Bob answers, Alice turns the receive ratchet to decrypt the message of Bob.

$$dk_0 = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], B, A)$$

- When Bob sends multiple messages, Alice turns the decryption ratchet.
- To separate the domains of the sending and receiving ratchets, we encode the sender and the receiver as input to the first KDF.



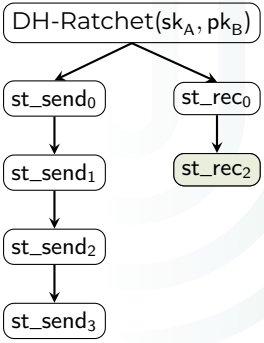
# Double Ratchet in Action

## View of Alice When Bob Answers

- When Bob answers, Alice turns the receive ratchet to decrypt the message of Bob.

$$dk_0 = \mathsf{KDF}([\mathsf{sk}_\mathsf{A}, \mathsf{pk}_\mathsf{B}], \mathsf{B}, \mathsf{A})$$

- When Bob sends multiple messages, Alice turns the decryption ratchet.
- To separate the domains of the sending and receiving ratchets, we encode the sender and the receiver as input to the first KDF.



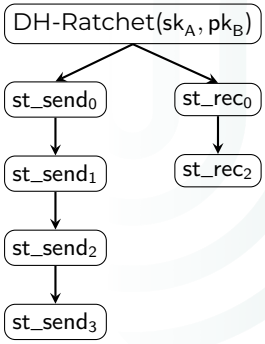
# Double Ratchet in Action

## View of Alice When Bob Answers

- When Bob answers, Alice turns the receive ratchet to decrypt the message of Bob.

$$dk_0 = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], \mathrm{B}, \mathrm{A})$$

- When Bob sends multiple messages, Alice turns the decryption ratchet.
- To separate the domains of the sending and receiving ratchets, we encode the sender and the receiver as input to the first KDF. A, B → B, A ←



# Double Ratchet in Action

## Turning the DH Ratchet

- Eventually, Alice and Bob want to achieve post-compromise security after the current session.
- Alice sends a new ephemeral public key to Bob and Bob returns with an ephemeral public key.
- They both update their states accordingly:

$$st_A = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], [\mathsf{esk}_A, \mathsf{epk}_B]), \mathsf{sk}_A$$
$$st\_send_A = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], [\mathsf{esk}_A, \mathsf{epk}_B], \mathsf{A}, \mathsf{B})$$
$$st\_rec_A = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], [\mathsf{esk}_A, \mathsf{epk}_B], \mathsf{B}, \mathsf{A})$$

- Both can start sending and receiving messages by turning a new symmetric ratchet.

$\mathsf{DH\text{-}Ratchet}(\mathsf{sk}_A, \mathsf{pk}_B)$



TU WIEN PRIVACY ENHANCING TECHNOLOGIES

# Double Ratchet in Action

## Turning the DH Ratchet

- Eventually, Alice and Bob want to achieve post-compromise security after the current session.
- Alice sends a new ephemeral public key to Bob and Bob returns with an ephemeral public key.
- They both update their states accordingly:

$$st_A = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], [\mathsf{esk}_A, \mathsf{epk}_B]), \mathsf{sk}_A$$
$$st\_send_A = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], [\mathsf{esk}_A, \mathsf{epk}_B], A, B)$$
$$st\_rec_A = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], [\mathsf{esk}_A, \mathsf{epk}_B], B, A)$$

- Both can start sending and receiving messages by turning a new symmetric ratchet.

$\mathsf{DH\text{-}Ratchet}(\mathsf{sk}_A, \mathsf{esk}_A, \mathsf{epk}_B, \mathsf{pk}_B)$



TU WIEN PRIVACY ENHANCING TECHNOLOGIES

# Double Ratchet in Action

## Turning the DH Ratchet

- Eventually, Alice and Bob want to achieve post-compromise security after the current session.
- Alice sends a new ephemeral public key to Bob and Bob returns with an ephemeral public key.
- They both update their states accordingly:

$$st_A = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], [\mathsf{esk}_A, \mathsf{epk}_B]), \mathsf{sk}_A$$
$$st\_send_A = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], [\mathsf{esk}_A, \mathsf{epk}_B], \mathsf{A}, \mathsf{B})$$
$$st\_rec_A = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], [\mathsf{esk}_A, \mathsf{epk}_B], \mathsf{B}, \mathsf{A})$$

$\mathsf{DH\text{-}Ratchet}(\mathsf{sk}_A, \mathsf{esk}_A, \mathsf{epk}_B, \mathsf{pk}_B)$

- Both can start sending and receiving messages by turning a new symmetric ratchet.



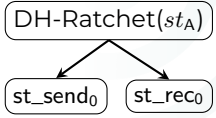
TU WIEN PRIVACY ENHANCING TECHNOLOGIES

# Double Ratchet in Action

## Sending and Receiving

- Alice has the state

$$st_A = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], [\mathsf{esk}_A, \mathsf{epk}_B]), \mathsf{sk}_A$$
$$st\_send_A = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], [\mathsf{esk}_A, \mathsf{epk}_B], A, B)$$
$$st\_rec_A = \mathsf{KDF}([\mathsf{sk}_A, \mathsf{pk}_B], [\mathsf{esk}_A, \mathsf{epk}_B], B, A)$$




- Alice can start sending messages to Bob by turning the symmetric ratchet.

$$ek_0, st\_send'_A = \mathsf{KDF}(st\_send_A)$$

- Alice can start receiving messages from Bob by turning the symmetric ratchet.

$$dk_0, st\_rec'_A = \mathsf{KDF}(st\_rec_A)$$

# Double Ratchet in Action

## Sending and Receiving

- Alice has the state

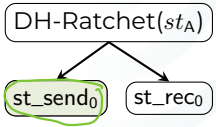
$$st_{\mathsf{A}} = \mathsf{KDF}([\mathsf{sk}_{\mathsf{A}}, \mathsf{pk}_{\mathsf{B}}], [\mathsf{esk}_{\mathsf{A}}, \mathsf{epk}_{\mathsf{B}}]), \mathsf{sk}_{\mathsf{A}}$$
$$st\_send_{\mathsf{A}} = \mathsf{KDF}([\mathsf{sk}_{\mathsf{A}}, \mathsf{pk}_{\mathsf{B}}], [\mathsf{esk}_{\mathsf{A}}, \mathsf{epk}_{\mathsf{B}}], \mathsf{A}, \mathsf{B})$$
$$st\_rec_{\mathsf{A}} = \mathsf{KDF}([\mathsf{sk}_{\mathsf{A}}, \mathsf{pk}_{\mathsf{B}}], [\mathsf{esk}_{\mathsf{A}}, \mathsf{epk}_{\mathsf{B}}], \mathsf{B}, \mathsf{A})$$

- Alice can start sending messages to Bob by turning the symmetric ratchet.

$$ek_0, st\_send'_{\mathsf{A}} = \mathsf{KDF}(st\_send_{\mathsf{A}})$$

- Alice can start receiving messages from Bob by turning the symmetric ratchet.

$$dk_0, st\_rec'_{\mathsf{A}} = \mathsf{KDF}(st\_rec_{\mathsf{A}})$$



# Double Ratchet in Action

## Sending and Receiving

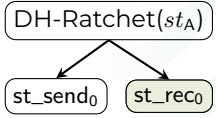
- Alice has the state

$$st_{\mathsf{A}} = \mathsf{KDF}([\mathsf{sk}_{\mathsf{A}}, \mathsf{pk}_{\mathsf{B}}], [\mathsf{esk}_{\mathsf{A}}, \mathsf{epk}_{\mathsf{B}}]), \mathsf{sk}_{\mathsf{A}}$$

$$st\_send_{\mathsf{A}} = \mathsf{KDF}([\mathsf{sk}_{\mathsf{A}}, \mathsf{pk}_{\mathsf{B}}], [\mathsf{esk}_{\mathsf{A}}, \mathsf{epk}_{\mathsf{B}}], \mathsf{A}, \mathsf{B})$$

$$st\_rec_{\mathsf{A}} = \mathsf{KDF}([\mathsf{sk}_{\mathsf{A}}, \mathsf{pk}_{\mathsf{B}}], [\mathsf{esk}_{\mathsf{A}}, \mathsf{epk}_{\mathsf{B}}], \mathsf{B}, \mathsf{A})$$

$\mathsf{epk}_{\mathsf{B}}$




- Alice can start sending messages to Bob by turning the symmetric ratchet.

$$ek_0, st\_send'_{\mathsf{A}} = \mathsf{KDF}(st\_send_{\mathsf{A}})$$

- Alice can start receiving messages from Bob by turning the symmetric ratchet.

$$dk_0, st\_rec'_{\mathsf{A}} = \mathsf{KDF}(st\_rec_{\mathsf{A}})$$



TU WIEN
PRIVACY ENHANCING TECHNOLOGIES

# Sources


- Moxie Marlinspike, Trevor Perrin. *The Double Ratchet Algorithm*. Signal Protocol Documentation.
- Hugo Krawczyk. *Cryptographic Extraction and Key Derivation: The HKDF Scheme*. 2010.
- Whitfield Diffie, Martin E. Hellman. *New Directions in Cryptography*. IEEE Transactions on Information Theory, 1976.




TU WIEN
PRIVACY ENHANCING TECHNOLOGIES

# Case Study: Threema Messaging

# Threema

## History and Overview

- **Founded in 2012:** By Manuel Kasper in Switzerland.
- **Privacy Focus:**
  - Registration without phone numbers or email addresses.
  - Strong commitment to user anonymity and privacy.
- **Encryption:** End-to-end encryption for messages, calls, and media.
- **Swiss Origin:** Leveraging Switzerland's strong privacy laws.



TU WIEN PRIVACY ENHANCING TECHNOLOGIES

# Encrypting Messages prior 2023

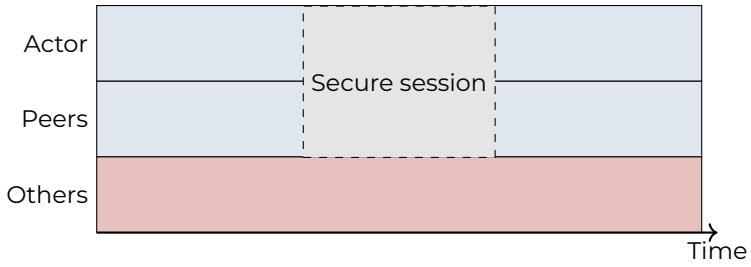
| **Alice**$(\mathsf{sk}_A, \mathsf{pk}_B)$ | | **Bob**$(\mathsf{sk}_B, \mathsf{pk}_A)$ |
|---|---|---|
| $\mathsf{K}_{A,B} \leftarrow \mathsf{DH}(\mathsf{sk}_A, \mathsf{pk}_B)$ | | $\mathsf{K}_{A,B} \leftarrow \mathsf{DH}(\mathsf{sk}_B, \mathsf{pk}_A)$ |
| $\mathsf{non} \leftarrow\!\!\$\ \{0,1\}^{128}$ | | |
| $\mathsf{nonces}_A \leftarrow \mathsf{nonces}_A \cup \{\mathsf{non}\}$ | | |
| $\mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathsf{K}_{A,B}, \mathsf{msg}, \mathsf{non})$ | | |
| | $\xrightarrow{\mathsf{ctxt}, \mathsf{non}}$ | |
| | | **if** $\mathsf{non} \in \mathsf{nonces}_B$ : discard |
| | | $msg \leftarrow \mathsf{Dec}(\mathsf{K}_{A,B}, \mathsf{ctxt}, \mathsf{non})$ |
| | | $\mathsf{nonces}_B \leftarrow \mathsf{nonces}_B \cup \mathsf{non}$ |



TU WIEN
PRIVACY ENHANCING TECHNOLOGIES

# No Forward Security

## Classical adversary model



# Threema Before 2023

## Encryption Model and Limitations

- **Encryption Scheme:**
  - A single Diffie-Hellman (DH) key exchange established a shared key.
  - The shared key was used for encrypting and decrypting all communications.
- **Limitations:**
  - **No Forward Security:**
    - ★ A compromised long-term key exposed all past messages.
  - **No Key Updates:**
    - ★ Static key usage increased vulnerability to long-term key compromise.



TU WIEN PRIVACY ENHANCING TECHNOLOGIES

# Threema: Analysis of a Secure Messenger

TCS ⟶ Oracle Padding Attacks

- In 2023, Kenneth G. Paterson, Matteo Scarlata, and Kien Tuong Truong, from ETH Zurich analyzed Threema and found a total of seven security vulnerabilities besides having no forward security and no post-compromise security.
- Threema proposed the Ibex protocol in December 2022 having now available also forward security.
- Ibex uses two steps of a DH ratchet until a single ephemeral key of each participant is part of the state.
- Besides this, a symmetric ratchet is used.
- 



TU WIEN
PRIVACY ENHANCING TECHNOLOGIES

# Feedback

Your *anonymous feedback* is invaluable to us, and we are truly grateful for your time and insights.

Thank you for helping us improve!

https://tuwel.tuwien.ac.at/mod/feedback/view.php?id=2459261



TU WIEN PRIVACY ENHANCING TECHNOLOGIES


PRIVACY
ENHANCING
TECHNOLOGIES