A Comprehensive Study of the Signal Handshake Protocol: Bundled Authenticated Key Exchange

Keitaro Hashimoto, AIST

Shuichi Katsumata, PQShield & AIST

Guilhem Niot, PQShield & Univ Rennes, CNRS, IRISA

Ida Tucker, PQShield

Thom Wiggers, PQShield

This presentation is based on results obtained from a project, JPNP24003, commissioned by the New Energy and Industrial Technology Development Organization (NEDO)

The Signal protocol

The Signal protocol powers messaging for billions of users:

- ➔ Signal app
- ➔ WhatsApp
- ➔ Google RCS
- ➔ Facebook Messenger

Relies on **handshake protocol** to set up **secure conversations**:

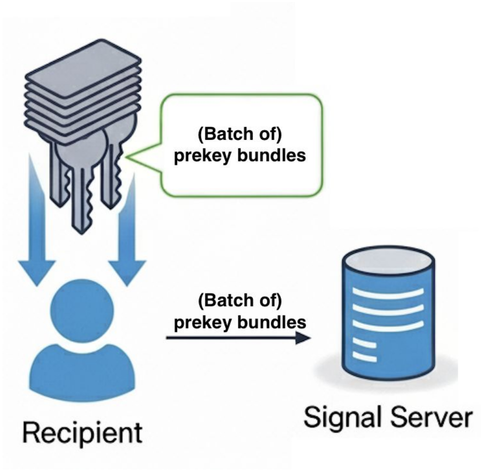
- ➔ X3DH (2016 – classically secure)
- ➔ PQXDH (2023 – HNDL secure)
- ➔ Fully Post-Quantum proposals: Bre+22; Col+24; Has+22

Choosing a fully PQ protocol requires comparing proposals.

Analyses use ad-hoc models tailored to individual protocols, making comparisons difficult.

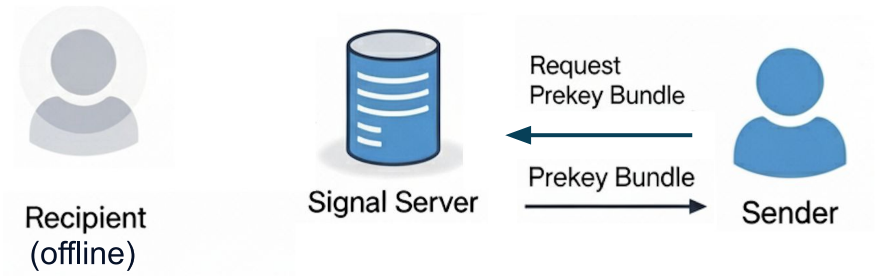
Prekey Bundles: where tailored models fall short

- Distinct component of Signal handshake protocols
- Users upload (batches of) key material onto the server



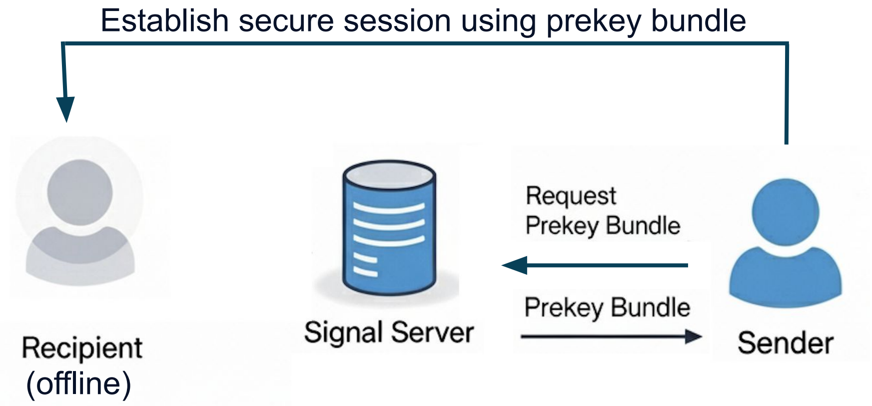
Prekey Bundles: where tailored models fall short

- Distinct component of Signal handshake protocols
- Users upload (batches of) key material onto the server
- Senders can establish communication even when recipients are offline.



Prekey Bundles: where tailored models fall short

- Distinct component of Signal handshake protocols
- Users upload (batches of) key material onto the server
- Senders can establish communication even when recipients are offline.



Prekey Bundles: where tailored models fall short

Existing analyses of Signal handshake protocols (ad-hoc tailored models):

- Treat prekey bundles differently in each model
- Do not fully treat uploading of prekey bundles
- Don't capture all relevant adversaries
- Attained security **hard to compare**

Contributions: framework for Signal handshake protocols

- Bundled AKE protocols (**BAKE**)
  - Formally model **prekey bundles** and their **states**
  - **Unified framework** for Signal handshake protocols
  - Establish various levels of **security**
  - Framework for analyzing **deniability**
    - Novel metric $\rightarrow$ relaxed, pragmatic guarantees for deniability
- RingXKEM: new efficient PQ Signal handshake protocol
  - Not captured by previous models
  - From Ring Signatures
- FalconRS: compact, post-quantum, and **deniable** RS from Falcon
  - Novel metric for **deniability** of RSs

Bundled Authenticated Key Exchange (BAKE)

Bundled Authenticated Key Exchange (BAKE)

Syntax

- ➔ $\mathrm{BAKE.IDKeyGen}(1^{\lambda}) \rightarrow (ik, isk)$ — identity key generation algorithm
- ➔ $\mathrm{BAKE.PreKeyBundleGen}(isk_u) \rightarrow (\overrightarrow{prek}, st_u)$ — prekey bundle generation algorithm
- ➔ $\mathrm{BAKE.Send}(isk_s, ik_r, prek_{r,t}) \rightarrow (K, \rho)$ — sender algorithm
- ➔ $\mathrm{BAKE.Receive}(isk_r, st_r, ik_s, t, \rho) \rightarrow (K', st_r)$ — (deterministic) receiver algorithm

Bundled Authenticated Key Exchange (BAKE)

Syntax

- BAKE.IDKeyGen(1^λ) $\rightarrow$ (ik, isk) — identity key generation algorithm
- BAKE.PreKeyBundleGen(isk_u) $\rightarrow$ ($\overrightarrow{prek}$, st_u) — prekey bundle generation algorithm
- BAKE.Send(isk_s, ik_r, $prek_{r,t}$) $\rightarrow$ (K, ρ) — sender algorithm
- BAKE.Receive(isk_r, st_r, ik_s, t, ρ) $\rightarrow$ (K', st_r) — (deterministic) receiver algorithm

Single state for all uploaded prekey bundles

Bundled Authenticated Key Exchange (BAKE)

Syntax

- ➔ BAKE.IDKeyGen(1^λ) $\rightarrow$ (ik, isk) — identity key generation algorithm
- ➔ BAKE.PreKeyBundleGen(isk$_u$) $\rightarrow$ ($\overrightarrow{\text{prek}}$, st$_u$) — prekey bundle generation algorithm
- ➔ BAKE.Send(isk$_s$, ik$_r$, prek$_{r,t}$) $\rightarrow$ (K, ρ) — sender algorithm
- ➔ BAKE.Receive(isk$_r$, st$_r$, ik$_s$, t, ρ) $\rightarrow$ (K', st$_r$) — (deterministic) receiver algorithm

Refreshes prekey bundles and state

Ensuring availability: last resort prekeys

Batch of prekey bundles contains:

- **One-time** prekey bundles (prek_1 , prek_2 , … , prek_L)
  - **Used once –** prekey bundle and associated state **deleted after use**
- A single **last resort** prekey bundle $\mathrm{prek}_\perp$
  - **Used if one-time prekey bundles run out**
  - **Re-used** until the next call to PreKeyBundleGen

Consequence:

- Exchanges using **last-resort** prekey bundle are vulnerable to **state compromise**
  - even after the handshake completes
  - until the next call of PreKeyBundleGen

Correctness and Security of BAKE

Correctness

Users **honestly execute the BAKE protocol** → derive **identical session key.**

Security (game based)

- Extend **AKE** definitions to capture **(last-resort) prekey bundles**
- Model the security of a BAKE protocol via:
  - **Match soundness** game
    - Parties have a consistent view of who they are talking to
  - **Key indistinguishability** game
    - Established session keys look random to the adversary.

Deniability of BAKE: unified & modular framework

Setting:

- **Accuser** collects evidence relative to **accused user** which is provided to a **distinguisher**
- **Distinguisher** decides if evidence could have been simulated by the accuser

Modular framework:

- **Differentiate who provides information to the distinguisher** (accusers / accused)
- **Accuser capabilities : standard** (honest-but-curious) **/ strong** (malicious)
- **Classical / quantum accuser** and **distinguisher**
- **Scopes:**
  - **Local** – accuser is sender or receiver
  - **Global** – both sender and receiver deny participation

Deniability of BAKE: pragmatic metric

Prior Work:

- Real & simulated evidence **indistinguishable by D.**
  - Statistical distance between distributions is close
  - Conservative (simulator outputs evidence **with same probability** as accused user)

Our approach:

- Accused user only needs to prove that **evidence could have been simulated**
- Inspired by differential privacy and differential indistinguishability
- **Distributions close** in terms of **hockey-stick divergence**

RingXKEM

new efficient

fully PQ Signal handshake protocol

RingXKEM: birds eye view

- ➔ Inspired by deniable AKE protocol by Hashimoto et al. [Has+21; Has+22]
  - ◆ Based on Ring Signatures (RS)
- ➔ Extended to BAKE syntax
- ➔ Optimization using Merkle trees:
  - ◆ ↘ Receiver bandwidth
  - ◆ ↘ Server storage

RingXKEM: without Merkle Tree Optimization

function $\mathsf{RingXKEM.IdKeyGen}(1^\lambda)$
    $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$
    $(\mathsf{rvk}, \mathsf{rsk}) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)$
    return $(\mathsf{ik} \coloneqq (\mathsf{ek}, \mathsf{rvk}), \mathsf{isk} \coloneqq (\mathsf{dk}, \mathsf{rsk}))$

function $\mathsf{RingXKEM.PreKeyBundleGen}(\mathsf{isk}_u)$
    $(\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$
    $D_{\mathsf{kem}}, D_{\rho_\perp} \coloneqq \emptyset$ ▷ *Initialize empty lists*
    for $t \in [L] \cup \{\perp\}$ **do**
        $(\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$
        $\sigma_{u,t} \xleftarrow{\$} \mathsf{RS.Sign}\,(\mathsf{rsk}_u, \widehat{\mathsf{ek}}_{u,t}, \{\, \mathsf{rvk}_u \,\})$
    $(\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)$ ▷ *Discard* rsk
    for $t \in [L] \cup \{\perp\}$ **do** ▷ *One-time prekey bundles*
        $\mathsf{prek}_{u,t} \coloneqq (\widehat{\mathsf{ek}}_{u,t}, \sigma_{u,t}, \mathsf{rvk})$
        $D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$
    return $\begin{pmatrix} \overrightarrow{\mathsf{prek}}_u \coloneqq (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}, \\ \mathsf{st}_u \coloneqq (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

RingXKEM: without Merkle Tree Optimization

function $\mathsf{RingXKEM.IdKeyGen}(1^\lambda)$
  $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$
  $(\mathsf{rvk}, \mathsf{rsk}) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)$
  return $(\mathsf{ik} \coloneqq (\mathsf{ek}, \mathsf{rvk}), \mathsf{isk} \coloneqq (\mathsf{dk}, \mathsf{rsk}))$

function $\mathsf{RingXKEM.PreKeyBundleGen}(\mathsf{isk}_u)$
  $(\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$
  $D_{\mathsf{kem}}, D_{\rho_\perp} \coloneqq \emptyset$ ▹ *Initialize empty lists*
  for $t \in [L] \cup \{\perp\}$ **do**
    $(\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$
    $\sigma_{u,t} \xleftarrow{\$} \mathsf{RS.Sign}\,(\mathsf{rsk}_u, \widehat{\mathsf{ek}}_{u,t}, \{\, \mathsf{rvk}_u \,\})$
  $(\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)$ ▹ *Discard* rsk
  for $t \in [L] \cup \{\perp\}$ **do** ▹ *One-time prekey bundles*
    $\mathsf{prek}_{u,t} \coloneqq (\widehat{\mathsf{ek}}_{u,t}, \sigma_{u,t}, \mathsf{rvk})$
    $D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$
  return $\begin{pmatrix} \overrightarrow{\mathsf{prek}}_u \coloneqq (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}, \\ \mathsf{st}_u \coloneqq (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

RingXKEM: without Merkle Tree Optimization

function $\mathsf{RingXKEM.IdKeyGen}(1^\lambda)$
- $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)$
- $(\mathsf{rvk}, \mathsf{rsk}) \xleftarrow{\$} \mathsf{RS.KeyGen}(1^\lambda)$
- **return** $(\mathsf{ik} \coloneqq (\mathsf{ek}, \mathsf{rvk}), \mathsf{isk} \coloneqq (\mathsf{dk}, \mathsf{rsk}))$

function $\mathsf{RingXKEM.PreKeyBundleGen}(\mathsf{isk}_u)$
- $(\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$
- $D_{\mathsf{kem}}, D_{\rho_\perp} \coloneqq \emptyset$ ▷ *Initialize empty lists*
- **for** $t \in [L] \cup \{\perp\}$ **do**
  - $\boxed{(\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)}$
  - $\sigma_{u,t} \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_u, \widehat{\mathsf{ek}}_{u,t}, \{\mathsf{rvk}_u\})$
- $(\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}(1^\lambda)$ ▷ *Discard* rsk
- **for** $t \in [L] \cup \{\perp\}$ **do** ▷ *One-time prekey bundles*
  - $\mathsf{prek}_{u,t} \coloneqq (\boxed{\widehat{\mathsf{ek}}_{u,t}}, \sigma_{u,t}, \mathsf{rvk})$
  - $D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$
- **return** $\begin{pmatrix} \overrightarrow{\mathsf{prek}}_u \coloneqq (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}, \\ \mathsf{st}_u \coloneqq (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

RingXKEM: without Merkle Tree Optimization

function RingXKEM.IdKeyGen(1^λ)
  $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)$
  $(\mathsf{rvk}, \mathsf{rsk}) \xleftarrow{\$} \mathsf{RS.KeyGen}(1^\lambda)$
  return $(\mathsf{ik} \coloneqq (\mathsf{ek}, \mathsf{rvk}), \mathsf{isk} \coloneqq (\mathsf{dk}, \mathsf{rsk}))$

function RingXKEM.PreKeyBundleGen(isk_u)
  $(\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$
  $D_{\mathsf{kem}}, D_{\rho_\perp} \coloneqq \emptyset$ ▹ *Initialize empty lists*
  for $t \in [L] \cup \{\perp\}$ **do**
    $(\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)$
    $\boxed{\sigma_{u,t} \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_u, \widehat{\mathsf{ek}}_{u,t}, \{\mathsf{rvk}_u\})}$
  $(\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}(1^\lambda)$ ▹ *Discard rsk*
  for $t \in [L] \cup \{\perp\}$ **do** ▹ *One-time prekey bundles*
    $\mathsf{prek}_{u,t} \coloneqq (\widehat{\mathsf{ek}}_{u,t}, \boxed{\sigma_{u,t},} \mathsf{rvk})$
    $D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$
  return $\begin{pmatrix} \vec{\mathsf{prek}}_u \coloneqq (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}, \\ \mathsf{st}_u \coloneqq (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

RingXKEM: without Merkle Tree Optimization

$\textbf{function}\ \mathsf{RingXKEM.IdKeyGen}(1^\lambda)$

$\quad (\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$

$\quad (\mathsf{rvk}, \mathsf{rsk}) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)$

$\quad \textbf{return}\ (\mathsf{ik} \coloneqq (\mathsf{ek}, \mathsf{rvk}), \mathsf{isk} \coloneqq (\mathsf{dk}, \mathsf{rsk}))$

$\textbf{function}\ \mathsf{RingXKEM.PreKeyBundleGen}(\mathsf{isk}_u)$

$\quad (\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$

$\quad D_{\mathsf{kem}}, D_{\rho_\perp} \coloneqq \emptyset$ ▷ *Initialize empty lists*

$\quad \textbf{for}\ t \in [L] \cup \{\perp\}\ \textbf{do}$

$\quad\quad (\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$

$\quad\quad \sigma_{u,t} \xleftarrow{\$} \mathsf{RS.Sign}\,(\mathsf{rsk}_u, \widehat{\mathsf{ek}}_{u,t}, \{\mathsf{rvk}_u\})$

$\quad \boxed{(\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)\ \triangleright\ \textit{Discard}\ \mathsf{rsk}}$

$\quad \textbf{for}\ t \in [L] \cup \{\perp\}\ \textbf{do}$ ▷ *One-time prekey bundles*

$\quad\quad \mathsf{prek}_{u,t} \coloneqq (\widehat{\mathsf{ek}}_{u,t}, \sigma_{u,t}, \boxed{\mathsf{rvk}})$

$\quad\quad D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$

$\quad \textbf{return}\ \begin{pmatrix} \vec{\mathsf{prek}}_u \coloneqq (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}, \\ \mathsf{st}_u \coloneqq (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

RingXKEM: without Merkle Tree Optimization

$\textbf{function } \mathsf{RingXKEM.IdKeyGen}(1^\lambda)$

$\quad (\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)$

$\quad (\mathsf{rvk}, \mathsf{rsk}) \xleftarrow{\$} \mathsf{RS.KeyGen}(1^\lambda)$

$\quad \textbf{return } (\mathsf{ik} := (\mathsf{ek}, \mathsf{rvk}), \mathsf{isk} := (\mathsf{dk}, \mathsf{rsk}))$

$\textbf{function } \mathsf{RingXKEM.PreKeyBundleGen}(\mathsf{isk}_u)$

$\quad (\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$

$\quad D_{\mathsf{kem}}, D_{\rho_\perp} := \emptyset$ ▹ *Initialize empty lists*

$\quad \textbf{for } t \in [L] \cup \{\perp\} \textbf{ do}$

$\qquad (\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)$

$\qquad \sigma_{u,t} \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_u, \widehat{\mathsf{ek}}_{u,t}, \{\mathsf{rvk}_u\})$

$\quad (\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}(1^\lambda)$ ▹ *Discard* rsk

$\quad \textbf{for } t \in [L] \cup \{\perp\} \textbf{ do}$ ▹ *One-time prekey bundles*

$\qquad \mathsf{prek}_{u,t} := (\widehat{\mathsf{ek}}_{u,t}, \sigma_{u,t}, \mathsf{rvk})$

$\qquad D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$

$\quad \textbf{return } \begin{pmatrix} \vec{\mathsf{prek}}_u := (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}, \\ \mathsf{st}_u := (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

$\textbf{function } \mathsf{RingXKEM.Send}(\mathsf{isk}_s, \mathsf{ik}_r, \mathsf{prek}_r)$

$\quad (\mathsf{dk}_s, \mathsf{rsk}_s) \leftarrow \mathsf{isk}_s; \boxed{(\mathsf{ek}_r,} \mathsf{rvk}_r) \leftarrow \mathsf{ik}_r$

$\quad \boxed{(\widehat{\mathsf{ek}}_r}, \sigma_r, \mathsf{rvk}) \leftarrow \mathsf{prek}_r$

$\quad \textbf{require } [\![\mathsf{RS.Verify}(\{\mathsf{rvk}_r\}, \widehat{\mathsf{ek}}_r, \sigma_r) = 1]\!]$

$\quad (\mathsf{ss}_r, \mathsf{ct}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\mathsf{ek}_r)$

$\quad (\widehat{\mathsf{ss}}_r, \widehat{\mathsf{ct}}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\widehat{\mathsf{ek}}_r)$

$\quad \mathsf{content} := \mathsf{ik}_s \| \mathsf{ik}_r \| \mathsf{prek}_r \| \mathsf{ct}_r \| \mathsf{ct}_r$

$\quad K \| K_{\mathsf{ske}} := \mathsf{KDF}(\mathsf{ss}_r \| \widehat{\mathsf{ss}}_r, \mathsf{content})$

$\quad \sigma \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_s, \mathsf{content}, \{\mathsf{rvk}_s, \mathsf{rvk}\})$

$\quad \mathsf{ct}_{\mathsf{ske}} \xleftarrow{\$} \mathsf{SKE.Enc}(K_{\mathsf{ske}}, \sigma)$ ▹ *Mask ring sig*

$\quad \rho := (\mathsf{ct}_r, \widehat{\mathsf{ct}}_r, \mathsf{ct}_{\mathsf{ske}})$

$\quad \textbf{return } (K, \rho)$

RingXKEM: without Merkle Tree Optimization

function $\mathsf{RingXKEM.IdKeyGen}(1^\lambda)$
- $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$
- $(\mathsf{rvk}, \mathsf{rsk}) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)$
- **return** $(\mathsf{ik} := (\mathsf{ek}, \mathsf{rvk}), \mathsf{isk} := (\mathsf{dk}, \mathsf{rsk}))$

function $\mathsf{RingXKEM.PreKeyBundleGen}(\mathsf{isk}_u)$
- $(\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$
- $D_{\mathsf{kem}}, D_{\rho_\perp} := \emptyset$ ▷ *Initialize empty lists*
- **for** $t \in [L] \cup \{\perp\}$ **do**
  - $(\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$
  - $\sigma_{u,t} \xleftarrow{\$} \mathsf{RS.Sign}\,(\mathsf{rsk}_u, \widehat{\mathsf{ek}}_{u,t}, \{\,\mathsf{rvk}_u\,\})$
- $(\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)$ ▷ *Discard* rsk
- **for** $t \in [L] \cup \{\perp\}$ **do** ▷ *One-time prekey bundles*
  - $\mathsf{prek}_{u,t} := (\widehat{\mathsf{ek}}_{u,t}, \sigma_{u,t}, \mathsf{rvk})$
  - $D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$
- **return** $\begin{pmatrix} \overrightarrow{\mathsf{prek}}_u := (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}, \\ \mathsf{st}_u := (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

function $\mathsf{RingXKEM.Send}(\mathsf{isk}_s, \mathsf{ik}_r, \mathsf{prek}_r)$
- $(\mathsf{dk}_s, \boxed{\mathsf{rsk}_s)} \leftarrow \mathsf{isk}_s;\ (\mathsf{ek}_r, \mathsf{rvk}_r) \leftarrow \mathsf{ik}_r$
- $(\widehat{\mathsf{ek}}_r, \sigma_r, \boxed{\mathsf{rvk})} \leftarrow \mathsf{prek}_r$
- **require** $[\![\mathsf{RS.Verify}(\{\,\mathsf{rvk}_r\,\}, \widehat{\mathsf{ek}}_r, \sigma_r) = 1]\!]$
- $(\mathsf{ss}_r, \mathsf{ct}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\mathsf{ek}_r)$
- $(\widehat{\mathsf{ss}}_r, \widehat{\mathsf{ct}}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\widehat{\mathsf{ek}}_r)$
- $\mathsf{content} := \mathsf{ik}_s \| \mathsf{ik}_r \| \mathsf{prek}_r \| \mathsf{ct}_r \| \widehat{\mathsf{ct}}_r$
- $K \| K_{\mathsf{ske}} := \mathsf{KDF}(\mathsf{ss}_r \| \widehat{\mathsf{ss}}_r, \mathsf{content})$
- $\boxed{\sigma \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_s, \mathsf{content}, \{\,\mathsf{rvk}_s, \mathsf{rvk}\,\})}$
- $\mathsf{ct}_{\mathsf{ske}} \xleftarrow{\$} \mathsf{SKE.Enc}(K_{\mathsf{ske}}, \sigma)$ ▷ *Mask ring sig*
- $\rho := (\mathsf{ct}_r, \widehat{\mathsf{ct}}_r, \mathsf{ct}_{\mathsf{ske}})$
- **return** (K, ρ)

RingXKEM: without Merkle Tree Optimization

function $\mathsf{RingXKEM.IdKeyGen}(1^\lambda)$
- $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$
- $(\mathsf{rvk}, \mathsf{rsk}) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)$
- **return** $(\mathsf{ik} \coloneqq (\mathsf{ek}, \mathsf{rvk}), \mathsf{isk} \coloneqq (\mathsf{dk}, \mathsf{rsk}))$

function $\mathsf{RingXKEM.PreKeyBundleGen}(\mathsf{isk}_u)$
- $(\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$
- $D_{\mathsf{kem}}, D_{\rho_\perp} \coloneqq \emptyset$ ▹ *Initialize empty lists*
- **for** $t \in [L] \cup \{\perp\}$ **do**
  - $(\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$
  - $\sigma_{u,t} \xleftarrow{\$} \mathsf{RS.Sign}\,(\mathsf{rsk}_u, \widehat{\mathsf{ek}}_{u,t}, \{\,\mathsf{rvk}_u\,\})$
- $(\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)$ ▹ *Discard* rsk
- **for** $t \in [L] \cup \{\perp\}$ **do** ▹ *One-time prekey bundles*
  - $\mathsf{prek}_{u,t} \coloneqq (\widehat{\mathsf{ek}}_{u,t}, \sigma_{u,t}, \mathsf{rvk})$
  - $D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$
- **return** $\begin{pmatrix} \vec{\mathsf{prek}}_u \coloneqq (\mathsf{prek}_{u,t})_{t\in[L]\cup\{\perp\}}, \\ \mathsf{st}_u \coloneqq (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

function $\mathsf{RingXKEM.Send}(\mathsf{isk}_s, \mathsf{ik}_r, \mathsf{prek}_r)$
- $(\mathsf{dk}_s, \boxed{\mathsf{rsk}_s}) \leftarrow \mathsf{isk}_s;\ (\mathsf{ek}_r, \mathsf{rvk}_r) \leftarrow \mathsf{ik}_r$
- $(\widehat{\mathsf{ek}}_r, \sigma_r, \boxed{\mathsf{rvk})} \leftarrow \mathsf{prek}_r$
- **require** $[\![\mathsf{RS.Verify}(\{\,\mathsf{rvk}_r\,\}, \widehat{\mathsf{ek}}_r, \sigma_r) = 1]\!]$
- $(\mathsf{ss}_r, \mathsf{ct}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\mathsf{ek}_r)$
- $(\widehat{\mathsf{ss}}_r, \widehat{\mathsf{ct}}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\widehat{\mathsf{ek}}_r)$
- $\boxed{\mathsf{content} \coloneqq \mathsf{ik}_s \| \mathsf{ik}_r \| \mathsf{prek}_r \| \mathsf{ct}_r \| \widehat{\mathsf{ct}}_r}$
- $K \| K_{\mathsf{ske}} \coloneqq \mathsf{KDF}(\mathsf{ss}_r \| \widehat{\mathsf{ss}}_r, \mathsf{content})$
- $\boxed{\sigma \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_s, \mathsf{content}, \{\,\mathsf{rvk}_s, \mathsf{rvk}\,\})}$
- $\mathsf{ct}_{\mathsf{ske}} \xleftarrow{\$} \mathsf{SKE.Enc}(K_{\mathsf{ske}}, \sigma)$ ▹ *Mask ring sig*
- $\rho \coloneqq (\mathsf{ct}_r, \widehat{\mathsf{ct}}_r, \mathsf{ct}_{\mathsf{ske}})$
- **return** (K, ρ)

RingXKEM: without Merkle Tree Optimization

function $\mathsf{RingXKEM.IdKeyGen}(1^\lambda)$
- $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)$
- $(\mathsf{rvk}, \mathsf{rsk}) \xleftarrow{\$} \mathsf{RS.KeyGen}(1^\lambda)$
- **return** $(\mathsf{ik} := (\mathsf{ek}, \mathsf{rvk}), \mathsf{isk} := (\mathsf{dk}, \mathsf{rsk}))$

function $\mathsf{RingXKEM.PreKeyBundleGen}(\mathsf{isk}_u)$
- $(\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$
- $D_{\mathsf{kem}}, D_{\rho_\perp} := \emptyset$ ▹ *Initialize empty lists*
- **for** $t \in [L] \cup \{\perp\}$ **do**
  - $(\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)$
  - $\sigma_{u,t} \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_u, \widehat{\mathsf{ek}}_{u,t}, \{\mathsf{rvk}_u\})$
- $(\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}(1^\lambda)$ ▹ *Discard* rsk
- **for** $t \in [L] \cup \{\perp\}$ **do** ▹ *One-time prekey bundles*
  - $\mathsf{prek}_{u,t} := (\widehat{\mathsf{ek}}_{u,t}, \sigma_{u,t}, \mathsf{rvk})$
  - $D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$
- **return** $\begin{pmatrix} \vec{\mathsf{prek}}_u := (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}, \\ \mathsf{st}_u := (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

function $\mathsf{RingXKEM.Send}(\mathsf{isk}_s, \mathsf{ik}_r, \mathsf{prek}_r)$
- $(\mathsf{dk}_s, \mathsf{rsk}_s) \leftarrow \mathsf{isk}_s$; $(\mathsf{ek}_r, \mathsf{rvk}_r) \leftarrow \mathsf{ik}_r$
- $(\widehat{\mathsf{ek}}_r, \sigma_r, \mathsf{rvk}) \leftarrow \mathsf{prek}_r$
- **require** $[\![\mathsf{RS.Verify}(\{\mathsf{rvk}_r\}, \widehat{\mathsf{ek}}_r, \sigma_r) = 1]\!]$
- $(\mathsf{ss}_r, \mathsf{ct}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\mathsf{ek}_r)$
- $(\widehat{\mathsf{ss}}_r, \widehat{\mathsf{ct}}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\widehat{\mathsf{ek}}_r)$
- $\mathsf{content} := \mathsf{ik}_s \| \mathsf{ik}_r \| \mathsf{prek}_r \| \mathsf{ct}_r \| \widehat{\mathsf{ct}}_r$
- $\boxed{K \| K_{\mathsf{ske}} := \mathsf{KDF}(\mathsf{ss}_r \| \widehat{\mathsf{ss}}_r, \mathsf{content})}$
- $\sigma \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_s, \mathsf{content}, \{\mathsf{rvk}_s, \mathsf{rvk}\})$
- $\mathsf{ct}_{\mathsf{ske}} \xleftarrow{\$} \mathsf{SKE.Enc}(K_{\mathsf{ske}}, \sigma)$ ▹ *Mask ring sig.*
- $\rho := (\mathsf{ct}_r, \widehat{\mathsf{ct}}_r, \mathsf{ct}_{\mathsf{ske}})$
- **return** (K, ρ)

RingXKEM: without Merkle Tree Optimization

function RingXKEM.IdKeyGen(1^λ)
- $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$
- $(\mathsf{rvk}, \mathsf{rsk}) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)$
- **return** $(\mathsf{ik} := (\mathsf{ek}, \mathsf{rvk}), \mathsf{isk} := (\mathsf{dk}, \mathsf{rsk}))$

function RingXKEM.PreKeyBundleGen(isk_u)
- $(\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$
- $D_{\mathsf{kem}}, D_{\rho_\perp} := \emptyset$ ▹ *Initialize empty lists*
- **for** $t \in [L] \cup \{\perp\}$ **do**
  - $(\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$
  - $\sigma_{u,t} \xleftarrow{\$} \mathsf{RS.Sign}\,(\mathsf{rsk}_u, \widehat{\mathsf{ek}}_{u,t}, \{\,\mathsf{rvk}_u\,\})$
- $(\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)$ ▹ *Discard* rsk
- **for** $t \in [L] \cup \{\perp\}$ **do** ▹ *One-time prekey bundles*
  - $\mathsf{prek}_{u,t} := (\widehat{\mathsf{ek}}_{u,t}, \sigma_{u,t}, \mathsf{rvk})$
  - $D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$
- **return** $\begin{pmatrix} \overrightarrow{\mathsf{prek}}_u := (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}, \\ \mathsf{st}_u := (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

function RingXKEM.Send($\mathsf{isk}_s, \mathsf{ik}_r, \mathsf{prek}_r$)
- $(\mathsf{dk}_s, \mathsf{rsk}_s) \leftarrow \mathsf{isk}_s$; $(\mathsf{ek}_r, \mathsf{rvk}_r) \leftarrow \mathsf{ik}_r$
- $(\widehat{\mathsf{ek}}_r, \sigma_r, \mathsf{rvk}) \leftarrow \mathsf{prek}_r$
- **require** $[\![\mathsf{RS.Verify}(\{\,\mathsf{rvk}_r\,\}, \widehat{\mathsf{ek}}_r, \sigma_r) = 1]\!]$
- $(\mathsf{ss}_r, \mathsf{ct}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\mathsf{ek}_r)$
- $(\widehat{\mathsf{ss}}_r, \widehat{\mathsf{ct}}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\widehat{\mathsf{ek}}_r)$
- $\mathsf{content} := \mathsf{ik}_s \| \mathsf{ik}_r \| \mathsf{prek}_r \| \mathsf{ct}_r \| \widehat{\mathsf{ct}}_r$
- $K \| K_{\mathsf{ske}} := \mathsf{KDF}(\mathsf{ss}_r \| \widehat{\mathsf{ss}}_r, \mathsf{content})$
- $\sigma \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_s, \mathsf{content}, \{\,\mathsf{rvk}_s, \mathsf{rvk}\,\})$
- $\boxed{\mathsf{ct}_{\mathsf{ske}} \xleftarrow{\$} \mathsf{SKE.Enc}(K_{\mathsf{ske}}, \sigma)}$ ▹ *Mask ring sig*
- $\rho := (\mathsf{ct}_r, \widehat{\mathsf{ct}}_r, \mathsf{ct}_{\mathsf{ske}})$
- **return** (K, ρ)

RingXKEM: without Merkle Tree Optimization

function $\mathsf{RingXKEM.IdKeyGen}(1^\lambda)$
- $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$
- $(\mathsf{rvk}, \mathsf{rsk}) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)$
- **return** $(\mathsf{ik} := (\mathsf{ek}, \mathsf{rvk}), \mathsf{isk} := (\mathsf{dk}, \mathsf{rsk}))$

function $\mathsf{RingXKEM.PreKeyBundleGen}(\mathsf{isk}_u)$
- $(\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$
- $D_{\mathsf{kem}}, D_{\rho_\perp} := \emptyset$ ▹ *Initialize empty lists*
- **for** $t \in [L] \cup \{\perp\}$ **do**
  - $(\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda)$
  - $\sigma_{u,t} \xleftarrow{\$} \mathsf{RS.Sign}\,(\mathsf{rsk}_u, \widehat{\mathsf{ek}}_{u,t}, \{\,\mathsf{rvk}_u\,\})$
- $(\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}\,(1^\lambda)$ ▹ *Discard* rsk
- **for** $t \in [L] \cup \{\perp\}$ **do** ▹ *One-time prekey bundles*
  - $\mathsf{prek}_{u,t} := (\widehat{\mathsf{ek}}_{u,t}, \sigma_{u,t}, \mathsf{rvk})$
  - $D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$
- **return** $\begin{pmatrix} \vec{\mathsf{prek}}_u := (\mathsf{prek}_{u,t})_{t\in[L]\cup\{\perp\}}, \\ \mathsf{st}_u := (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

function $\mathsf{RingXKEM.Send}(\mathsf{isk}_s, \mathsf{ik}_r, \mathsf{prek}_r)$
- $(\mathsf{dk}_s, \mathsf{rsk}_s) \leftarrow \mathsf{isk}_s$; $(\mathsf{ek}_r, \mathsf{rvk}_r) \leftarrow \mathsf{ik}_r$
- $(\widehat{\mathsf{ek}}_r, \sigma_r, \mathsf{rvk}) \leftarrow \mathsf{prek}_r$
- **require** $[\![\mathsf{RS.Verify}(\{\,\mathsf{rvk}_r\,\}, \widehat{\mathsf{ek}}_r, \sigma_r) = 1]\!]$
- $(\mathsf{ss}_r, \mathsf{ct}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\mathsf{ek}_r)$
- $(\widehat{\mathsf{ss}}_r, \widehat{\mathsf{ct}}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\widehat{\mathsf{ek}}_r)$
- $\mathsf{content} := \mathsf{ik}_s \| \mathsf{ik}_r \| \mathsf{prek}_r \| \mathsf{ct}_r \| \widehat{\mathsf{ct}}_r$
- $K \| K_{\mathsf{ske}} := \mathsf{KDF}(\mathsf{ss}_r \| \widehat{\mathsf{ss}}_r, \mathsf{content})$
- $\sigma \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_s, \mathsf{content}, \{\,\mathsf{rvk}_s, \mathsf{rvk}\,\})$
- $\mathsf{ct}_{\mathsf{ske}} \xleftarrow{\$} \mathsf{SKE.Enc}(K_{\mathsf{ske}}, \sigma)$ ▹ *Mask ring sig*
- $\boxed{\rho := (\mathsf{ct}_r, \widehat{\mathsf{ct}}_r, \mathsf{ct}_{\mathsf{ske}})}$
- **return** (K, ρ)

RingXKEM: the Merkle Tree Optimization

$\textbf{function}\ \mathsf{RingXKEM.PreKeyBundleGen}(\mathsf{isk}_u)$

$\quad (\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$

$\quad D_{\mathsf{kem}}, D_{\rho_\perp} := \emptyset$ ▷ *Initialize empty lists*

$\quad \textbf{for}\ t \in [L] \cup \{\perp\}\ \textbf{do}$

$\quad\quad (\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \overset{\$}{\leftarrow} \mathsf{KEM.KeyGen}(1^\lambda)$

$\quad\quad \boxed{\sigma_{u,t} \overset{\$}{\leftarrow} \mathsf{RS.Sign}(\mathsf{rsk}_u, \widehat{\mathsf{ek}}_{u,t}, \{\mathsf{rvk}_u\})}$

$\quad (\mathsf{rvk}, \_) \overset{\$}{\leftarrow} \mathsf{RS.KeyGen}(1^\lambda)$ ▷ *Discard* rsk

$\quad \textbf{for}\ t \in [L] \cup \{\perp\}\ \textbf{do}$

$\quad\quad \mathsf{prek}_{u,t} := (\widehat{\mathsf{ek}}_{u,t}, \boxed{\sigma_{u,t}}, \mathsf{rvk})$

$\quad\quad D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$

$\quad \textbf{return} \begin{pmatrix} \vec{\mathsf{prek}}_u := \boxed{(\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}}, \\ \mathsf{st}_u := (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

Observation

- ➔ Users upload $L+1$ ring signatures to server
- ➔ **PQ** (ring) signatures are **big**
- ➔ Prekey bundles are **large**

RingXKEM: the Merkle Tree Optimization

function RingXKEM.PreKeyBundleGen(isk_u)

$(\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$

$D_{\mathsf{kem}}, D_{\rho_\perp} := \emptyset$ ▹ *Initialize empty lists*

for $t \in [L] \cup \{\perp\}$ **do**

$\quad (\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)$

$\boxed{\begin{array}{l} \triangleright \textit{ Create and sign Merkle tree} \\ (\mathsf{root}_u, \mathsf{tree}_u) \leftarrow \mathsf{MerkleTree}((\widehat{\mathsf{ek}}_{u,t})_{t \in [L] \cup \{\perp\}}) \\ \sigma_{u,\mathsf{root}} \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_u, \mathsf{root}_u, \{\mathsf{rvk}_u\}) \end{array}}$

$(\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}(1^\lambda)$ ▹ *Discard* rsk

for $t \in [L]$ **do** ▹ *One-time prekey bundles*

$\quad \mathsf{path}_{u,t} \leftarrow \mathsf{getMerklePath}(\mathsf{tree}_u, t)$

$\quad \mathsf{prek}_{u,t} := (\widehat{\mathsf{ek}}_{u,t}, \mathsf{path}_{u,t}, \boxed{\mathsf{root}_u, \sigma_{u,\mathsf{root}},} \mathsf{rvk})$

$\quad D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$

▹ *Last-resort prekey bundle* $t = \perp$

$\mathsf{path}_{u,\perp} \leftarrow \mathsf{getMerklePath}(\mathsf{tree}_u, L+1)$

$\mathsf{prek}_{u,\perp} := (\widehat{\mathsf{ek}}_{u,\perp}, \mathsf{path}_{u,\perp}, \boxed{\mathsf{root}_u, \sigma_{u,\mathsf{root}},} \mathsf{rvk})$

$D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,\perp}, \widehat{\mathsf{dk}}_{u,\perp})$

return $\begin{pmatrix} \vec{\mathsf{prek}}_u := (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}, \\ \mathsf{st}_u := (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

Merkle tree optimization

➔ Only **upload a single signature**

- accumulate KEM keys $(\widehat{\mathsf{ek}}_{u,t})_{t \in [L] \cup \{\perp\}}$
- only sign digest root

RingXKEM: the Merkle Tree Optimization

function $\mathsf{RingXKEM.PreKeyBundleGen}(\mathsf{isk}_u)$

$(\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$

$D_{\mathsf{kem}}, D_{\rho_\perp} := \emptyset$ ▹ *Initialize empty lists*

for $t \in [L] \cup \{\perp\}$ **do**

$\quad (\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)$

▹ *Create and sign Merkle tree*

$(\mathsf{root}_u, \mathsf{tree}_u) \leftarrow \mathsf{MerkleTree}((\widehat{\mathsf{ek}}_{u,t})_{t \in [L] \cup \{\perp\}})$

$\sigma_{u,\mathsf{root}} \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_u, \mathsf{root}_u, \{\mathsf{rvk}_u\})$

$(\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}(1^\lambda)$ ▹ *Discard* rsk

for $t \in [L]$ **do** ▹ *One-time prekey bundles*

$\quad \mathsf{prek}_{u,t} := (\widehat{\mathsf{ek}}_{u,t}, \quad \mathsf{root}_u, \sigma_{u,\mathsf{root}}, \mathsf{rvk})$

$\quad D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \mathsf{dk}_{u,t})$

▹ *Last-resort prekey bundle $t = \perp$*

$\mathsf{prek}_{u,\perp} := (\widehat{\mathsf{ek}}_{u,\perp}, \quad \mathsf{root}_u, \sigma_{u,\mathsf{root}}, \mathsf{rvk})$

$D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,\perp}, \mathsf{dk}_{u,\perp})$

return $\begin{pmatrix} \vec{\mathsf{prek}}_u := (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}, \\ \mathsf{st}_u := (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

Merkle tree optimization

- ➔ Only **upload a single signature**
  - ◆ accumulate KEM keys $(\widehat{\mathsf{ek}}_{u,t})_{t \in [L] \cup \{\perp\}}$
  - ◆ only sign digest root
- ➔ Server can compute **path** on the fly
  - ◆ ↘ server storage

RingXKEM: Key Indistinguishability

Key Indistinguishability against quantum adversaries:

- If KDF is pseudorandom,
- KEM is IND-CCA secure,
- and **RS is unforgeable against quantum adversaries.**

RingXKEM: Deniability

sender's identity key

receiver's prekey bundle

- By properties of RS , σ := $\mathrm{RS.Sign}(rsk_s, \mathrm{content}, \{ rkv_s , \widehat{rvk_r} \})$ in handshake message:
  - **Authenticates** the sender (to receiver), RS unforgeable
  - Does not reveal **which** key was used to sign RS anonymous
    - Either sender or receiver could have signed

RS anonymous against quantum adversaries → Accuser and Distinguisher can be quantum

RingXKEM: Comparison to X3DH / PQXDH

RingXKEM

- **Stronger security guarantees** than X3DH / PQXDH
- At least **as deniable as PQXDH** (for setting of concern to Signal)

Full post-quantum security

Requires RS **unforgeable and anonymous** against **quantum adversaries.**

PQ Ring Signatures: Instantiating RingXKEM

- Ideally from **standardized** PQ signatures
- 🥺 Inefficient
  - PQ + **anonymous** + compact : **difficult** to construct

Deniable Ring Signatures

Deniability: Weakening anonymity

- **Anonymity** : signatures produced using two secret keys are **indistinguishable**.
- New relaxation: **Deniability**
    - Signature gives no **hard evidence** about which key was used to sign

Design deniable PQ RS schemes for small ring sizes

- From **Standardized Falcon [Pre+22, GJK24b] – FalconRS**
- From MAYO [Beu+24] - MayoRS

Deniable PQ Ring Signatures: FalconRS and MayoRS

Scheme	PK	2-RSig	Negligible anonymity
FalconRS (from Falcon-512)	897 B	1288 B	×
MayoRS$_1$ (from MAYO$_1$)	1168 B	650 B	×
MayoRS$_2$ (from MAYO$_2$)	5488 B	368 B	×
Gandalf [GJK24b]	896 B	1236 B	×
Raptor [LAZ19b]	900 B	2532 B	×
Calamari [BKP20]	64 B	3662 B	✓
DualRing-LB [Yue+21]	2496 B	3877 B	✓
Falafl [BKP20]	4096 B	30 016 B	✓

Scheme	Keygen	Sign 2-ring	Verify 2-ring
Falcon-512	6.2 Mc	0.74 Mc	0.04 Mc
MAYO$_1$	0.24 Mc	1.1 Mc	0.28 Mc
MAYO$_2$	0.65 Mc	1.5 Mc	0.16 Mc
Raptor	27.1 Mc	5 Mc	2 Mc
Calamari	119.5 Mc	46 581 Mc	41 250 Mc
Falafl	0.1 Mc	163 Mc	76 Mc

FalconRS:

- From **standardized Falcon**
- **Compact** + implementations **outperforming prior work**

Ongoing / Future work

Collaboration with Felix Günther[1], Vadim Lyubashevsky[1], and Rolfe Schmidt[2]

➔ Designing fully PQ handshake protocol for Signal

➔ Space/bandwidth optimizations for Falcon-based signatures and RSs

➔ Considering the use of RingXKEM for the next fully PQ Signal protocol

Other applications of RS where deniability is sufficient?

➔ Use *efficient* & *compact* FalconRS based on *standardized* Falcon

1. IBM Research Europe, Zurich
2. Signal Messenger

Full versions

"Bundled Authenticated Key Exchange: A Concrete Treatment of (Post-Quantum) Signal's Handshake Protocol." By Hashimoto, Katsumata, and Wiggers.
In: USENIX Security 2025
eprint.iacr.org/2025/040

"Comprehensive Deniability Analysis of Signal Handshake Protocols: X3DH, PQXDH to Fully Post-Quantum with Deniable Ring Signatures." By Katsumata, Niot, Tucker, Wiggers.
In: USENIX Security 2025
eprint.iacr.org/2025/1090