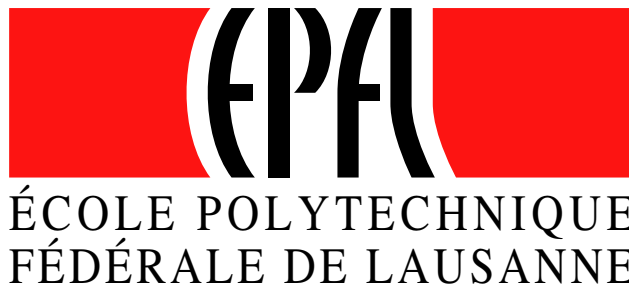
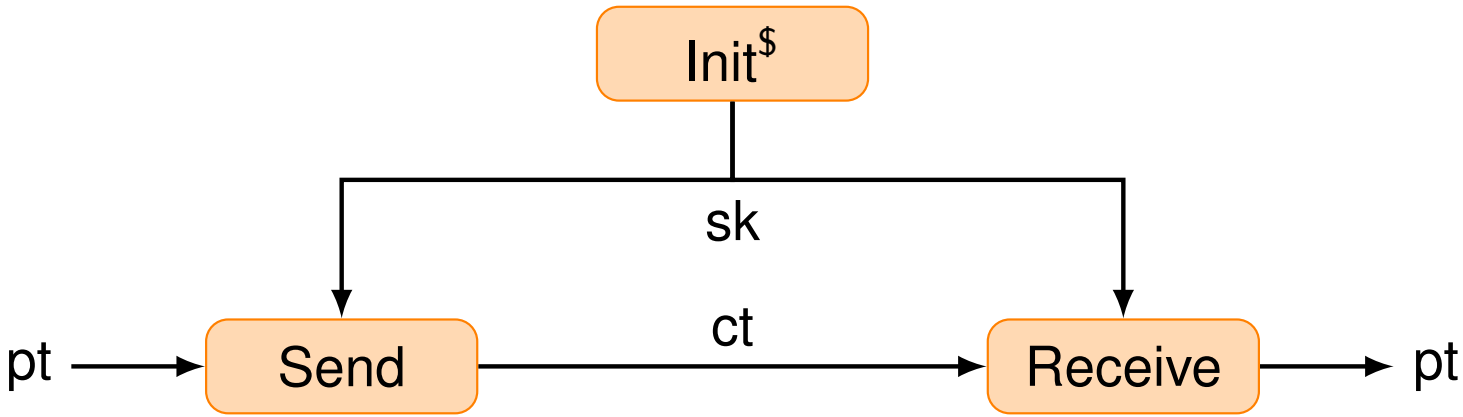
Secure Communication by Ratcheting

F Betül Durak and Serge Vaudenay



LASEC

Atomic Secure Communication

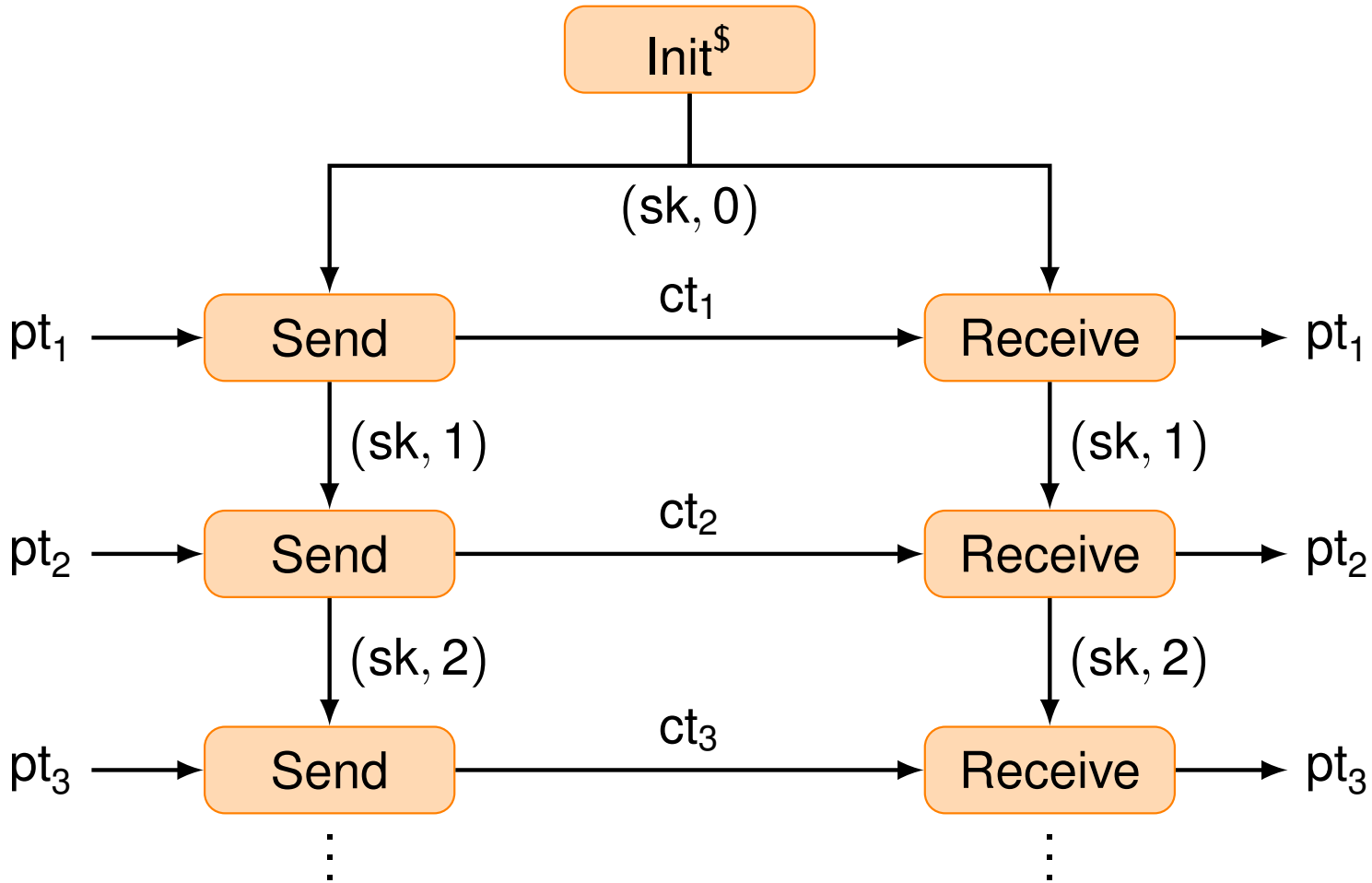


- the state is a (symmetric) secret key sk; it is FIXED
- Init $\xrightarrow{\$}$ sk

  Send(sk, pt) $\rightarrow$ ct

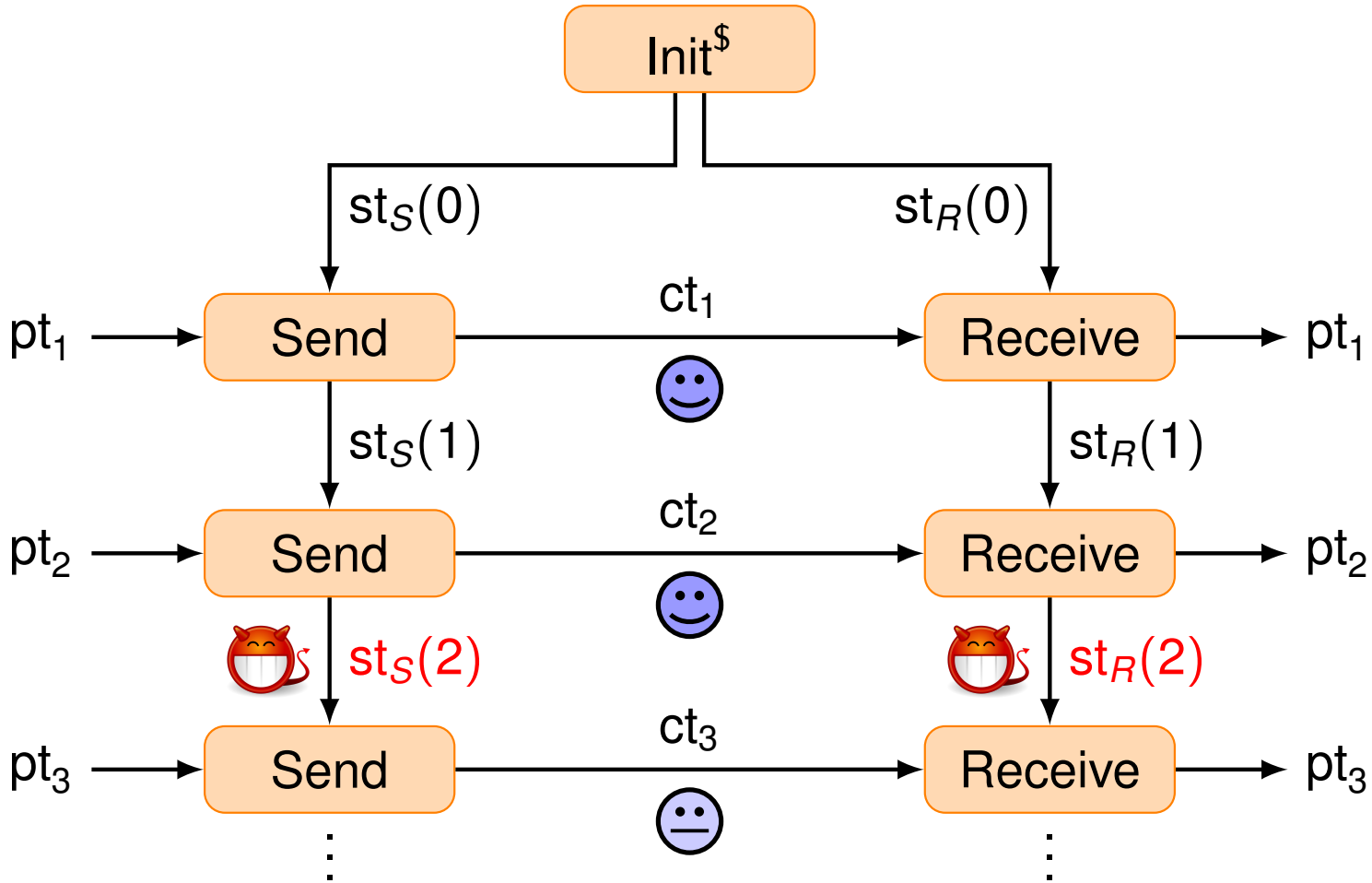
  Receive(sk, ct) $\rightarrow$ pt$'$
- **correctness**: pt $=$ pt$'$
- **security**: ...check the literature on AE...

Aim: Communication Integrity



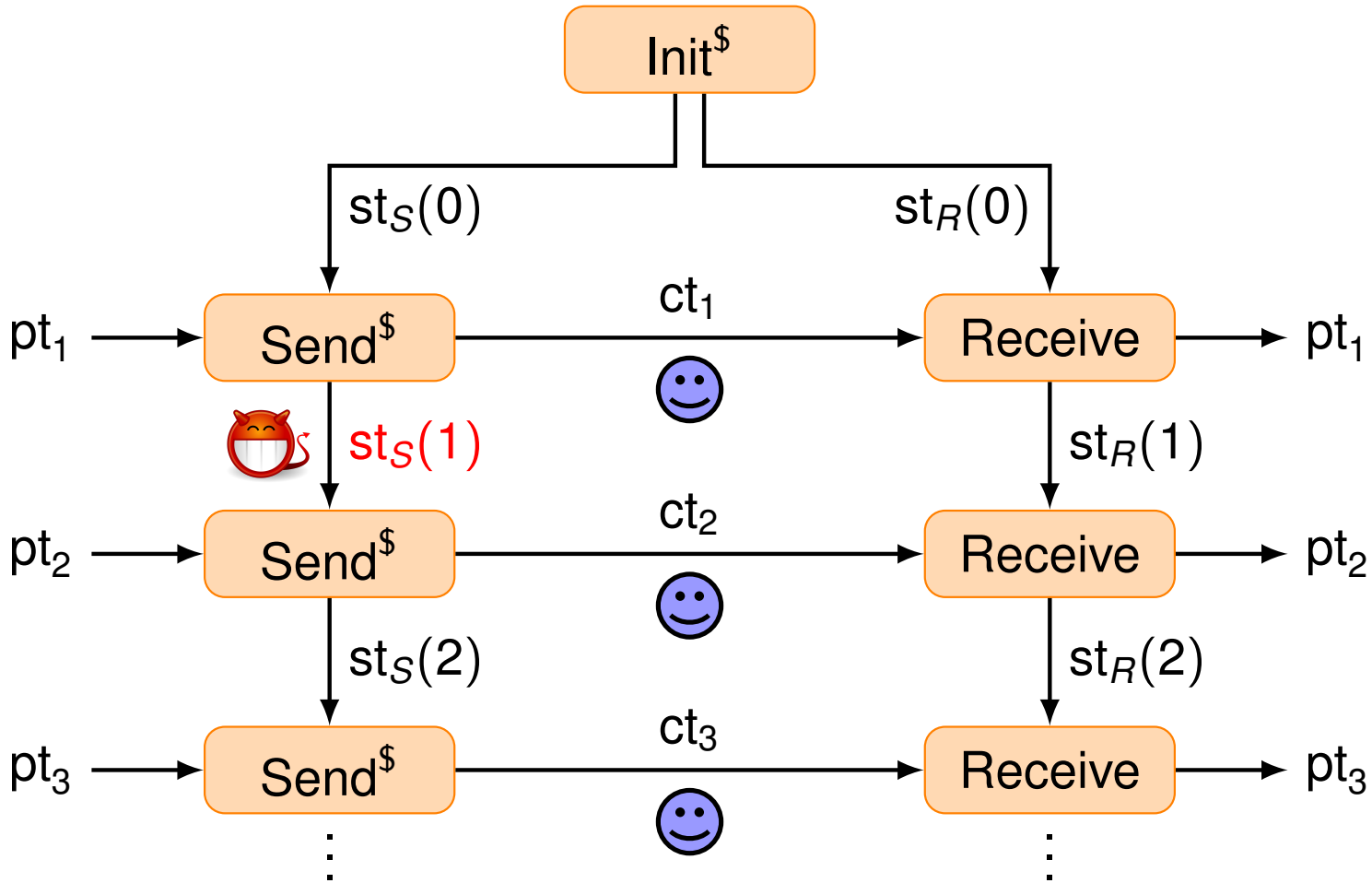
- **security**: the list of received messages can only be (a prefix of) the list of sent messages

Aim: Forward Privacy



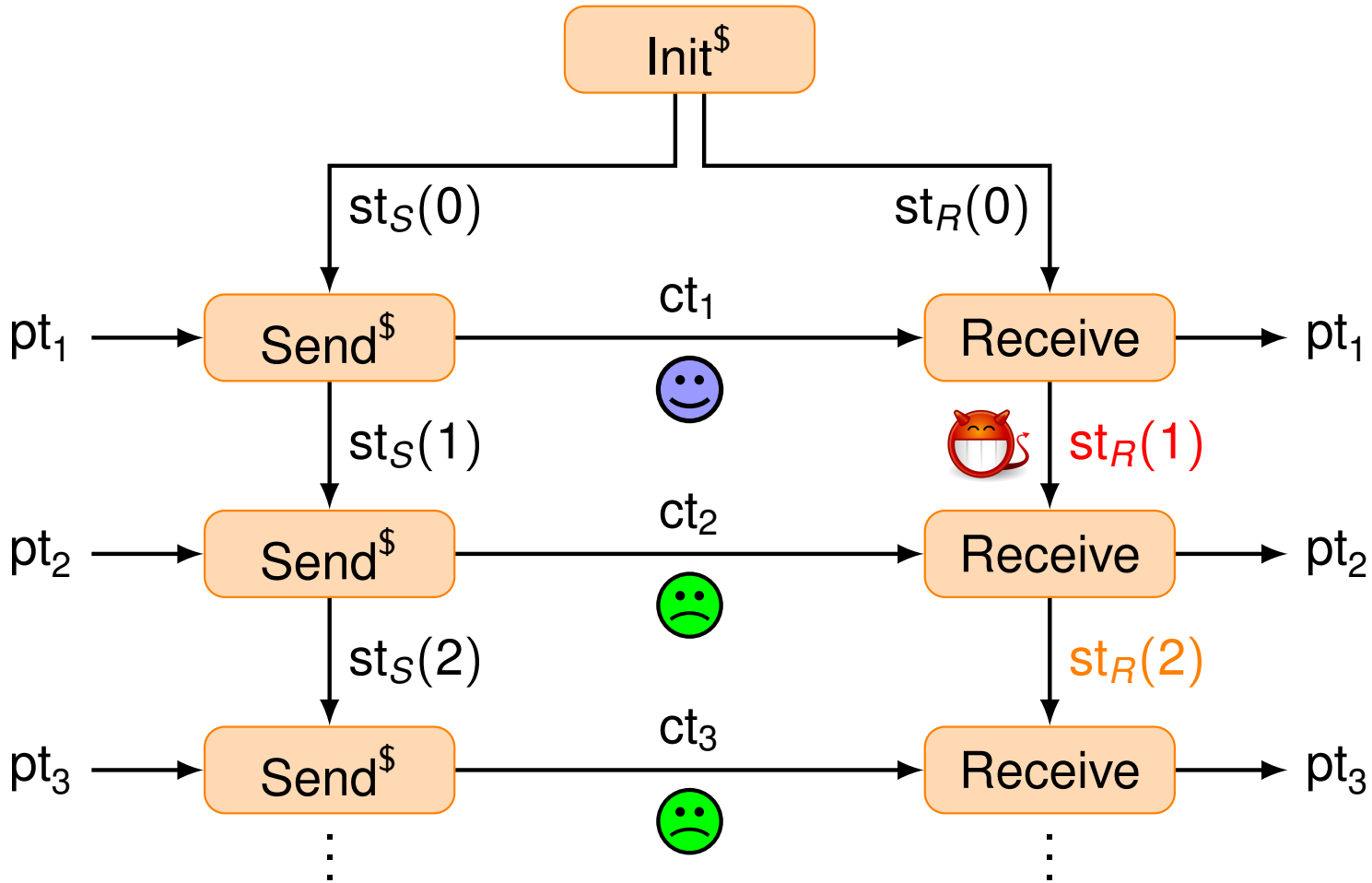
- **security**: leaking $\mathrm{st}_S(2), \mathrm{st}_R(2)$ does not compromise $\mathrm{pt}_1, \mathrm{pt}_2$

Aim: Post-Compromise Security (1)



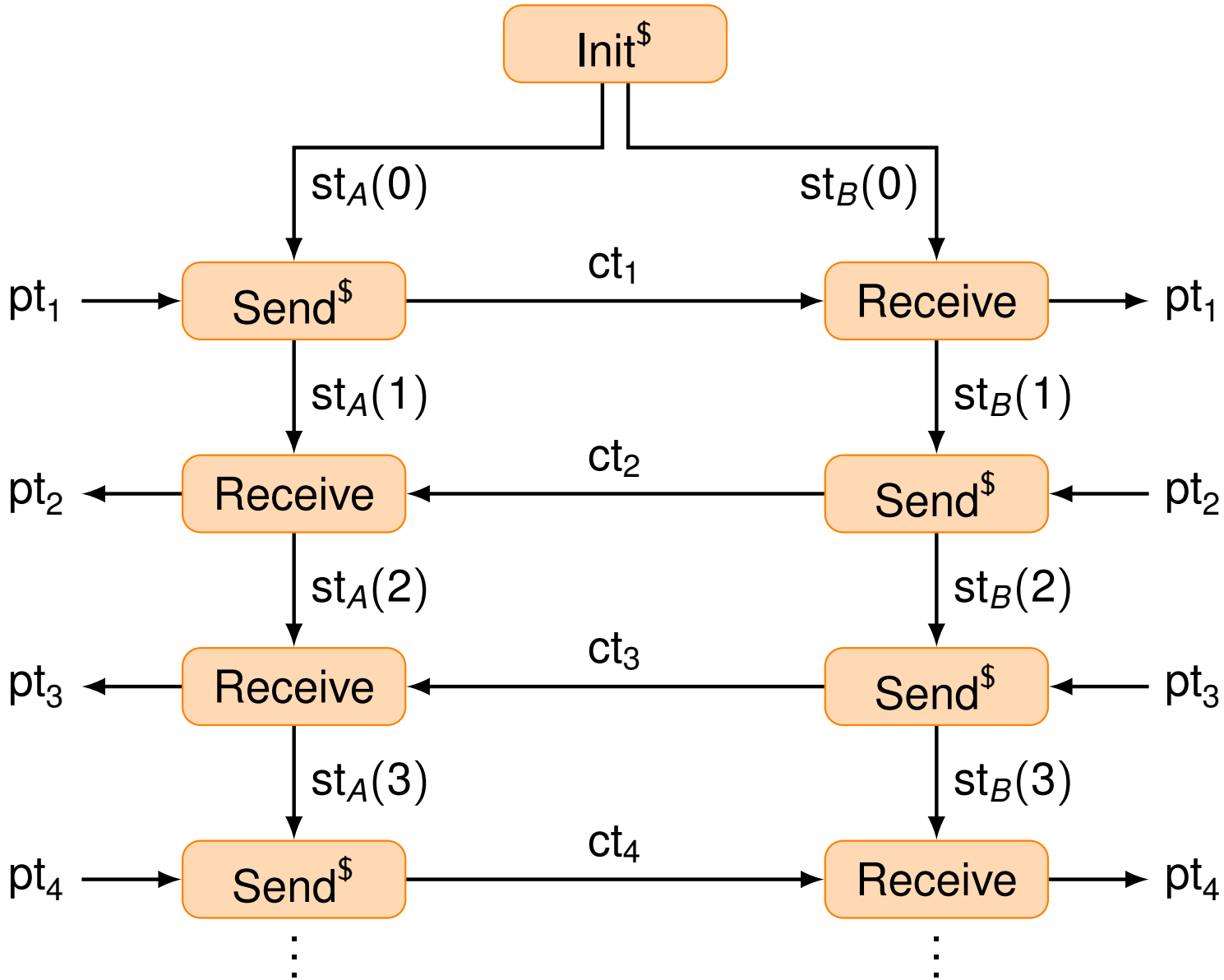
- **security**: leaking $st_S(1)$ does not compromise any message!

Aim: Post-Compromise Security (2)

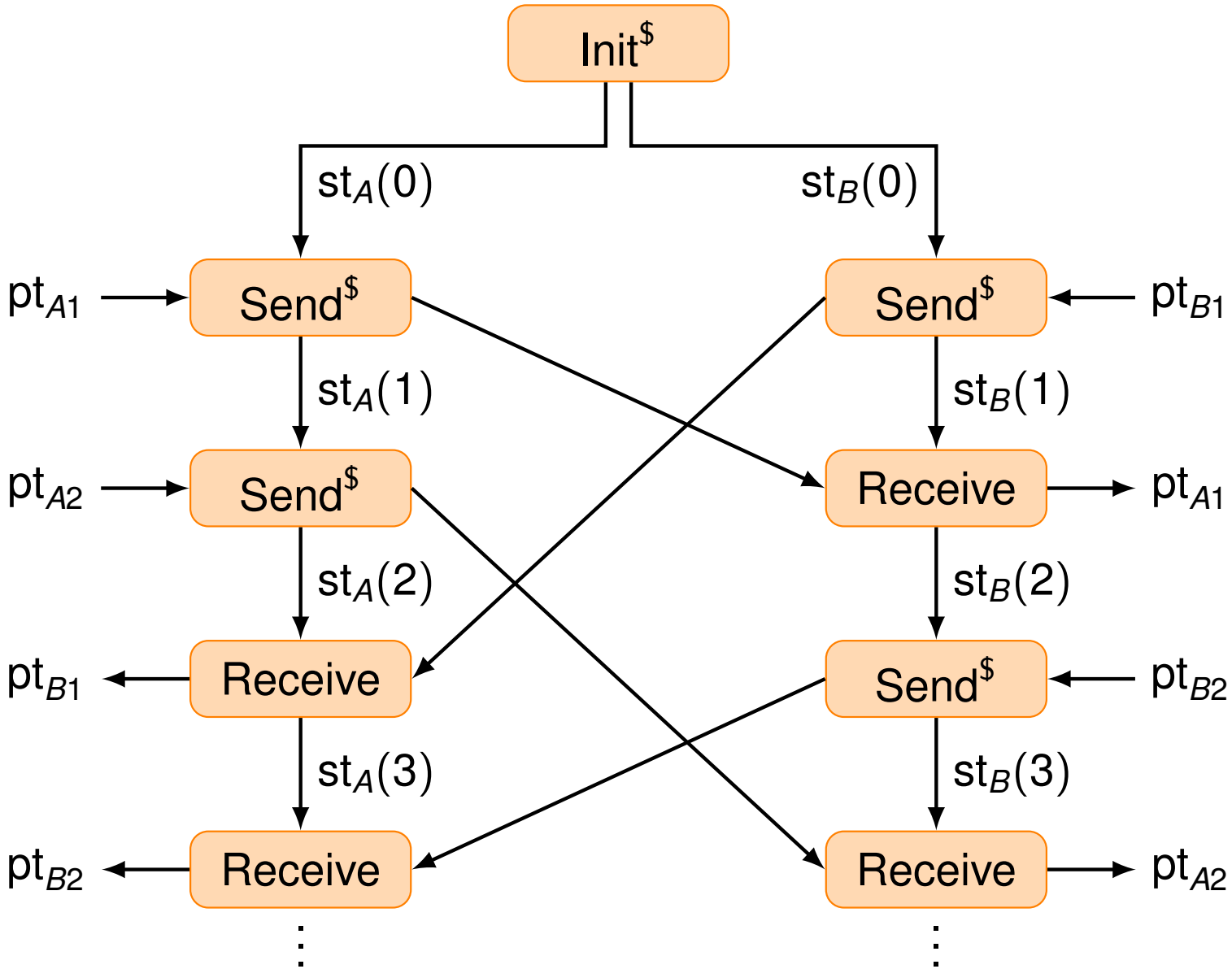


- **security**: leaking $\mathrm{st}_R(1)$ compromises $\mathrm{pt}_2, \mathrm{pt}_3$ (there is nothing to do about it)

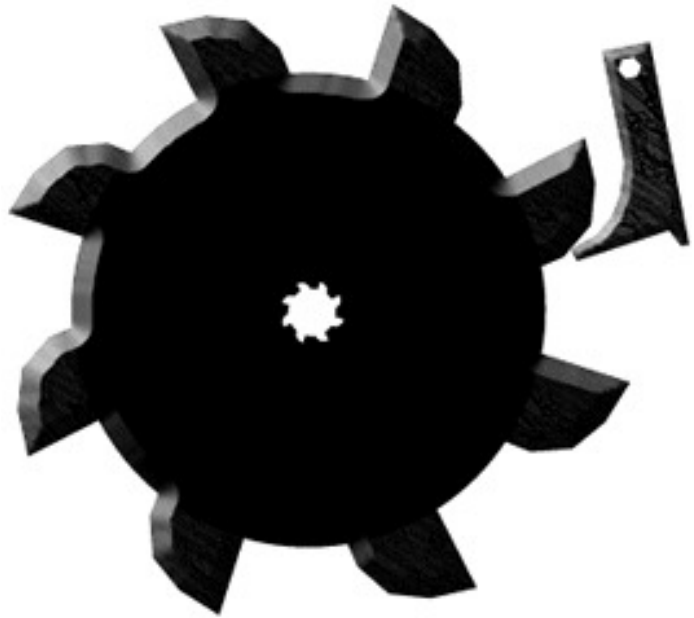
Aim: Bidirectional Communication



Aim: Asynchronous + Random Role



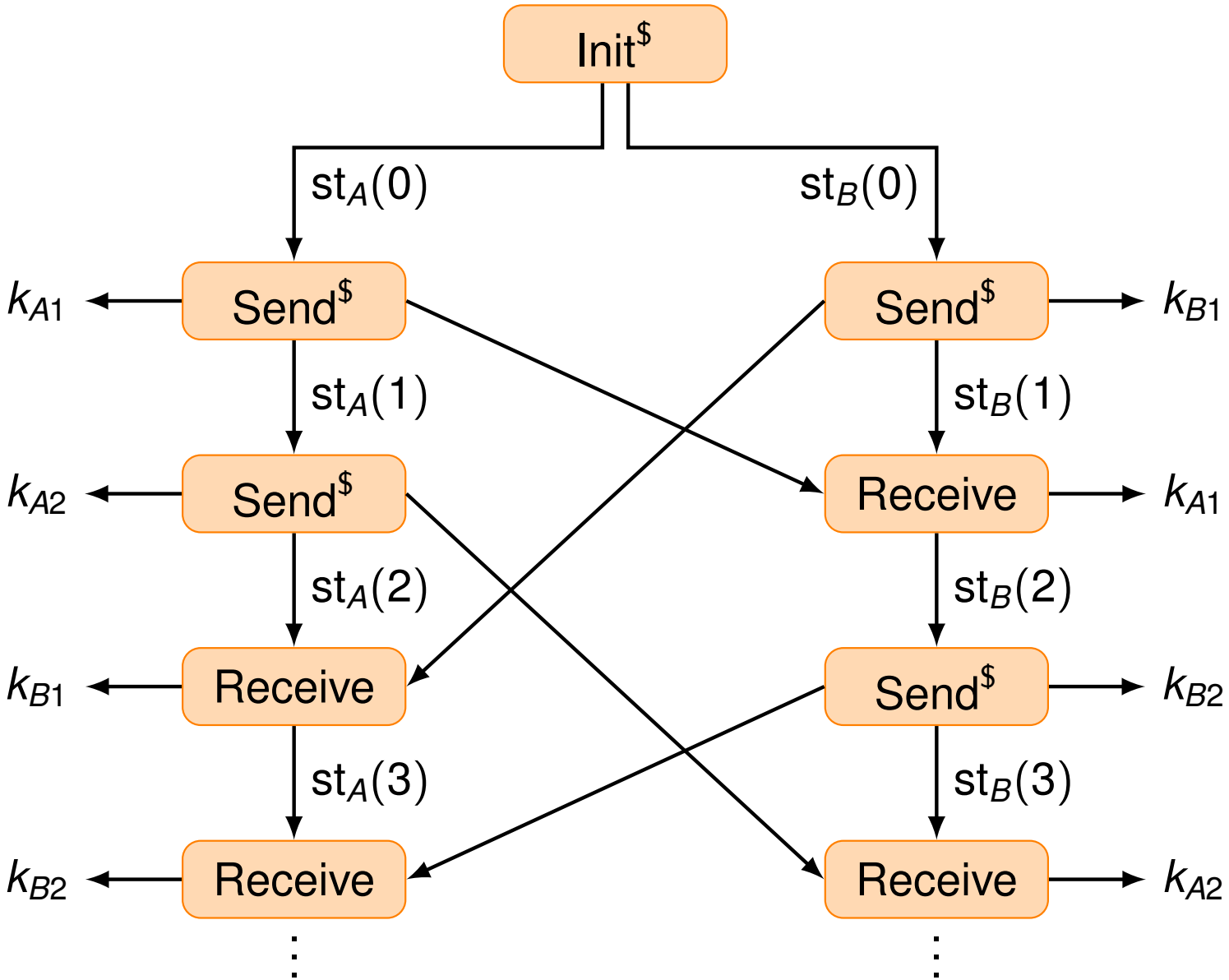
Ratchet



state update

- in a one-way manner (for **forward security**)
- using randomness (for **post-compromise security**)

Bidirectional Asynch. Ratcheted Key Agreement



CRYPTO 2017

Bellare-Singh-Asha-Jaeger-Nyayapati-Stepanovs
Ratcheted Encryption and Key Exchange: The Security of Messaging

- unidirectional
- no receiver leakage allowed
- complicated definitions

CRYPTO 2018

Poettering-Rösler
Ratcheted Key Exchange, Revisited

Jaeger-Stepanovs
Optimal Channel Security Against Fine-Grained State Compromise: The Safety of Messaging

- both need key update primitives (HIBE, random oracles, ...)
- complicated definitions

Poettering-Rösler Security Game

Game $\mathrm{KIND}^b_{\mathsf{R}}(\mathcal{A})$

00 For $u \in \{A, B\}$:
01 $\quad s_u \leftarrow 0$; $r_u \leftarrow 0$
02 $\quad e_u \leftarrow 0$; $\mathrm{EP}_u[\cdot] \leftarrow \bot$
03 $\quad E_u^{\vdash} \leftarrow 0$; $E_u^{\dashv} \leftarrow 0$
04 $\quad adc_u[\cdot] \leftarrow \bot$; $is_u \leftarrow \mathtt{T}$
05 $\quad key_u[\cdot] \leftarrow \bot$; $\mathrm{XP}_u \leftarrow \emptyset$
06 $\quad \mathrm{TR}_u \leftarrow \emptyset$; $\mathrm{CH}_u \leftarrow \emptyset$
07 $(S_A, S_B) \leftarrow_{\$} \mathrm{init}$
08 $b' \leftarrow_{\$} \mathcal{A}$
09 For $u \in \{A, B\}$:
10 $\quad$ Require $\mathrm{TR}_u \cap \mathrm{CH}_u = \emptyset$
11 Stop with b'

Oracle $\mathrm{Snd}(u, ad)$

12 Require $S_u \neq \bot$
13 $(S_u, k, c) \leftarrow_{\$} \mathrm{snd}(S_u, ad)$
14 If is_u:
15 $\quad adc_u[s_u] \leftarrow (ad, c)$
16 $\quad \mathrm{EP}_u[s_u] \leftarrow e_u$
17 $\quad E_u^{\dashv} \leftarrow E_u^{\dashv} + 1$
18 $key_u[\mathtt{S}, e_u, s_u] \leftarrow k$
19 $s_u \leftarrow s_u + 1$
20 Return c

Oracle $\mathrm{Reveal}(u, i)$

as in URKE/SRKE (Fig. 5)

Oracle $\mathrm{Rcv}(u, ad, c)$

21 Require $S_u \neq \bot$
22 If $is_u \wedge adc_{\bar{u}}[r_u] \neq (ad, c)$:
23 $\quad is_u \leftarrow \mathtt{F}$
24 $\quad$ If $r_u \in \mathrm{XP}_{\bar{u}}$:
25 $\quad\quad \mathrm{TR}_u \xleftarrow{\cup} \{\mathtt{S}\} \times \mathbb{N} \times [s_u, \ldots]$
26 $\quad\quad \mathrm{TR}_u \xleftarrow{\cup} \{\mathtt{R}\} \times \mathbb{N} \times [r_u, \ldots]$
27 If is_u:
28 $\quad E_u^{\vdash} \leftarrow \mathrm{EP}_{\bar{u}}[r_u]$
29 $\quad e_u \leftarrow e_u + 1$
30 $(S_u, k) \leftarrow \mathrm{rcv}(S_u, ad, c)$
31 If $S_u = \bot$: Return $\bot$
32 If is_u: $k \leftarrow \diamond$
33 $key_u[\mathtt{R}, E_u^{\vdash}, r_u] \leftarrow k$
34 $r_u \leftarrow r_u + 1$
35 Return

Oracle $\mathrm{Expose}(u)$

36 $\mathrm{TR}_u \xleftarrow{\cup} \{\mathtt{R}\} \times [E_u^{\vdash} \,..\, E_u^{\dashv}] \times [r_u, \ldots]$
37 If is_u:
38 $\quad \mathrm{XP}_u \xleftarrow{\cup} \{s_u\}$
39 $\quad \mathrm{TR}_{\bar{u}} \xleftarrow{\cup} \{\mathtt{S}\} \times [E_u^{\vdash} \,..\, E_u^{\dashv}] \times [r_u, \ldots]$
40 Return S_u

Oracle $\mathrm{Challenge}(u, i)$

as in URKE/SRKE (Fig. 5)

Jaeger-Stepanovs Security Game

Game $\mathrm{AEAC}^{\mathcal{D}}_{\mathsf{Ch}}$

$b \leftarrow\!\!\$\ \{0,1\}$; $s_{\mathcal{I}} \leftarrow 0$; $r_{\mathcal{I}} \leftarrow 0$; $s_{\mathcal{R}} \leftarrow 0$; $r_{\mathcal{R}} \leftarrow 0$
$\mathsf{restricted}_{\mathcal{I}} \leftarrow \mathsf{false}$; $\mathsf{restricted}_{\mathcal{R}} \leftarrow \mathsf{false}$
$\mathbf{forge}_{\mathcal{I}}[0 \ldots \infty] \leftarrow$ "nontrivial" ; $\mathbf{forge}_{\mathcal{R}}[0 \ldots \infty] \leftarrow$ "nontrivial"
$\mathcal{X}_{\mathcal{I}} \leftarrow 0$; $\mathcal{X}_{\mathcal{R}} \leftarrow 0$; $(st_{\mathcal{I}}, st_{\mathcal{R}}) \leftarrow\!\!\$\ \mathsf{Ch.Init}$
$(z_{\mathcal{I}}, z_{\mathcal{R}}) \leftarrow\!\!\$\ (\mathsf{Ch.SendRS})^2$; $(\eta_{\mathcal{I}}, \eta_{\mathcal{R}}) \leftarrow\!\!\$\ (\mathsf{Ch.RecvRS})^2$
$b' \leftarrow\!\!\$\ \mathcal{D}^{\textsc{Send},\textsc{Recv},\textsc{Exp}}$
Return $(b' = b)$

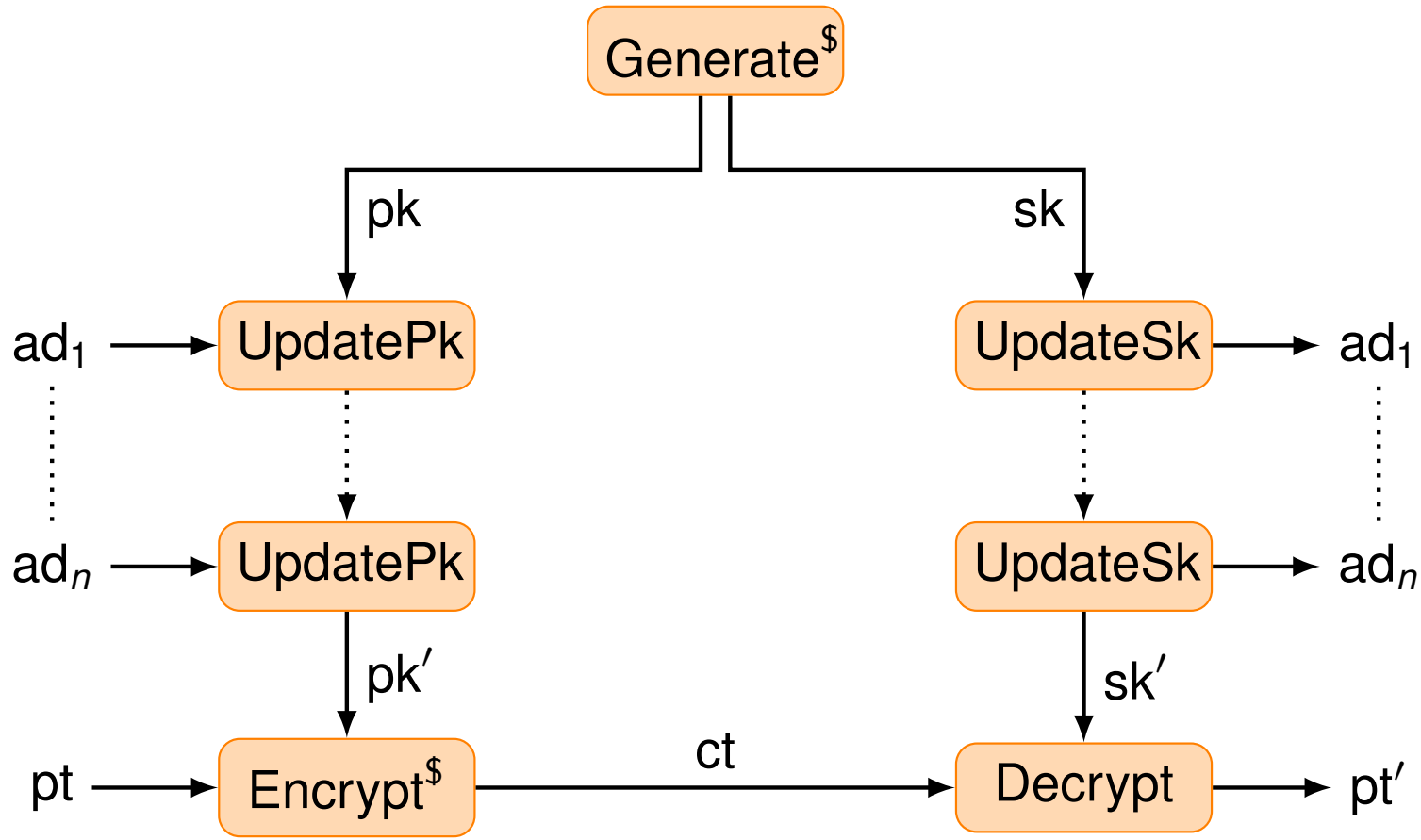
$\textsc{Send}(u, m_0, m_1, ad)$ // $u \in \{\mathcal{I}, \mathcal{R}\}, (m_0, m_1, ad) \in (\{0,1\}^*)^3$
If $\mathsf{nextop} \neq (u, \text{"send"})$ and $\mathsf{nextop} \neq \bot$ then return $\bot$
If $|m_0| \neq |m_1|$ then return $\bot$
If ($r_u < \mathcal{X}_u$ or $\mathsf{restricted}_u$ or $\mathbf{ch}_u[s_u + 1] =$ "forbidden") and $m_0 \neq m_1$ then
  Return $\bot$
$(st_u, c) \leftarrow \mathsf{Ch.Send}(st_u, ad, m_b; z_u)$
$\mathsf{nextop} \leftarrow \bot$; $s_u \leftarrow s_u + 1$; $z_u \leftarrow\!\!\$\ \mathsf{Ch.SendRS}$
If not $\mathsf{restricted}_u$ then $\mathbf{ctable}_{\overline{u}}[s_u] \leftarrow (c, ad)$
If $m_0 \neq m_1$ then $\mathbf{ch}_u[s_u] \leftarrow$ "done"
Return c

$\textsc{Recv}(u, c, ad)$ // $u \in \{\mathcal{I}, \mathcal{R}\}, (c, ad) \in (\{0,1\}^*)^2$
If $\mathsf{nextop} \neq (u, \text{"recv"})$ and $\mathsf{nextop} \neq \bot$ then return $\bot$
$(st_u, m) \leftarrow \mathsf{Ch.Recv}(st_u, ad, c; \eta_u)$
$\mathsf{nextop} \leftarrow \bot$; $\eta_u \leftarrow\!\!\$\ \mathsf{Ch.RecvRS}$
If $m = \bot$ then return $\bot$
$r_u \leftarrow r_u + 1$
If $\mathbf{forge}_u[r_u] =$ "trivial" and $(c, ad) \neq \mathbf{ctable}_u[r_u]$ then
  $\mathsf{restricted}_u \leftarrow \mathsf{true}$
If $\mathsf{restricted}_u$ or ($b = 0$ and $(c, ad) \neq \mathbf{ctable}_u[r_u]$) then
  Return m
Return $\bot$

$\textsc{Exp}(u, \mathsf{rand})$ // $u \in \{\mathcal{I}, \mathcal{R}\}, \mathsf{rand} \in \{\varepsilon, \text{"send"}, \text{"recv"}\}$
If $\mathsf{nextop} \neq \bot$ then return $\bot$
If $\mathsf{restricted}_u$ then return (st_u, z_u, η_u)
If $\exists i \in (r_u, s_{\overline{u}}]$ s.t. $\mathbf{ch}_{\overline{u}}[i] =$ "done" then
  Return $\bot$
$\mathbf{forge}_{\overline{u}}[s_u + 1] \leftarrow$ "trivial" ; $(z, \eta) \leftarrow (\varepsilon, \varepsilon)$; $\mathcal{X}_{\overline{u}} \leftarrow s_u + 1$
If $\mathsf{rand} =$ "send" then
  $\mathsf{nextop} \leftarrow (u, \text{"send"})$; $z \leftarrow z_u$; $\mathcal{X}_{\overline{u}} \leftarrow s_u + 2$
  $\mathbf{forge}_{\overline{u}}[s_u + 2] \leftarrow$ "trivial" ; $\mathbf{ch}_u[s_u + 1] \leftarrow$ "forbidden"
Else if $\mathsf{rand} =$ "recv" then
  $\mathsf{nextop} \leftarrow (u, \text{"recv"})$; $\eta \leftarrow \eta_u$
Return (st_u, z, η)

Cryptosystem with Key Update (for PR18 and JS18)

- Generate $\xrightarrow{\$}$ (sk, pk), Encrypt(pk, pt) $\xrightarrow{\$}$ ct, Decrypt(sk, ct) $\to$ pt′,
  UpdateSk(sk, ad) $\to$ sk′, UpdatePk(pk, ad) $\to$ pk′
- **correctness**: for all coins, pt, $\mathsf{ad}_1, \ldots, \mathsf{ad}_n$, if



then pt = pt′

- **security**: ...well...

Cryptosystem with Key Update from HIBE

Algorithm Generate()

1: HIBE.Init $\xrightarrow{\$}$ (msk, mpk)
2: pk $\leftarrow$ (mpk, $\perp$) ▷ second part is id (empty for root)
3: **return** (msk, pk) ▷ root secret is msk

Algorithm UpdateSk(sk, ad)

4: **return** HIBE.Extract(sk, ad) ▷ extract ad-child secret

Algorithm UpdatePk(pk, ad)

5: parse pk $=$ (mpk, id) ▷ mpk never changes
6: **return** (mpk, id.ad) ▷ just concatenate ad to id

Algorithm Encrypt(pk, pt)

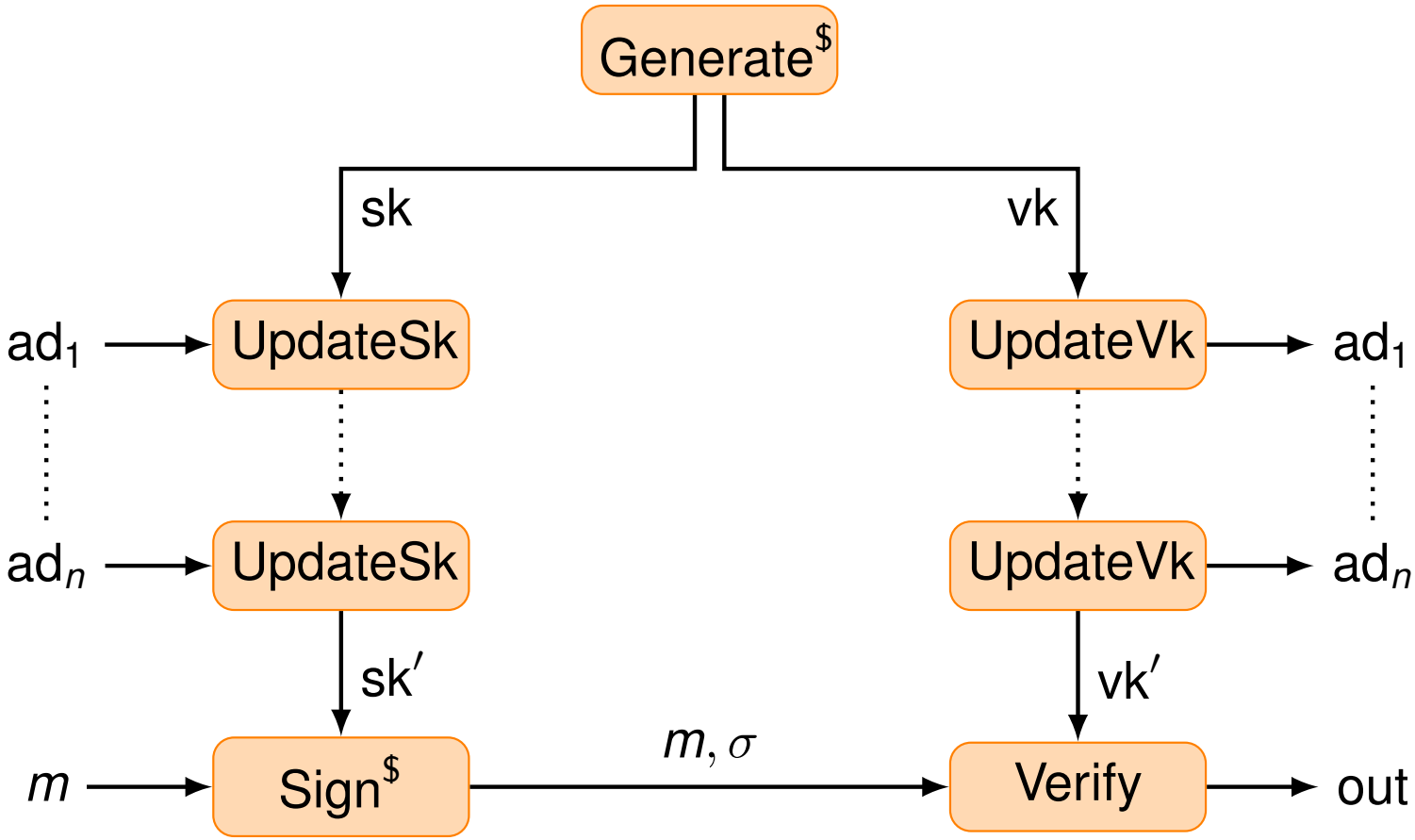
7: parse pk $=$ (mpk, id)
8: **return** HIBE.Encrypt(mpk, id, pt)

Algorithm Decrypt(sk, ct)

9: **return** HIBE.Decrypt(sk, ct)

Signature with Key Update (for JS18)

- Generate $\xrightarrow{\$}$ (sk, vk), Sign(sk, m) $\xrightarrow{\$}$ σ, Verify(vk, σ, m) $\rightarrow$ accept/reject,
  UpdateSk(sk, ad) $\rightarrow$ sk′, UpdatePk(vk, ad) $\rightarrow$ vk′
- **correctness**: for all coins, m, $\mathrm{ad}_1, \ldots, \mathrm{ad}_n$, if



  then out = accept
- **security**: ...well...

Our Definition

- 3 algorithms: $\text{Init}^{\$}$, $\text{Send}^{\$}$, Receive
- correctness
- KIND-security: generated keys look like random
  caveat: exceptions... (difficult to identify)
- FORGE-security: participants see the same messages
  caveat: “trivial forgeries” (after state exposure)
- RECOVER-security:
  after forgeries, genuine messages are no longer accepted

BARK

Bidirectional Asynchronous Ratcheted Key Agreement

Correctness

For all sequence sched, $\Pr[\text{Correctness}(\text{sched}) \to 1] = 0$

Oracle RATCH(P, rec, upd)

1: $(\text{acc}, \text{st}'_P, k_P) \leftarrow \text{Receive}(\text{st}_P, \text{upd})$
2: **if** acc **then**
3: $\quad \text{st}_P \leftarrow \text{st}'_P$
4: $\quad$ append k_P to $\text{received}^P_{\text{key}}$
5: **end if**
6: **return** acc

Oracle RATCH(P, send)

7: $(\text{st}'_P, \text{upd}, k_P) \leftarrow \text{Send}(\text{st}_P)$
8: $\text{st}_P \leftarrow \text{st}'_P$
9: append k_P to $\text{sent}^P_{\text{key}}$
10: **return** upd

Game Correctness(sched)

1: set all lists and queues to $\emptyset$
2: $\text{Init}(1^\lambda) \xrightarrow{\$} (\text{st}_A, \text{st}_B, z)$
3: $i \leftarrow 0$
4: **loop**
5: $\quad i \leftarrow i + 1$
6: $\quad (P, \text{role}) \leftarrow \text{sched}_i$
7: $\quad$ **if** role $=$ rec **then**
8: $\quad\quad$ **if** in_queue^P empty **then return** 0
9: $\quad\quad$ pull upd from in_queue^P
10: $\quad\quad$ acc $\leftarrow$ RATCH(P, rec, upd)
11: $\quad\quad$ **if** acc $=$ false **then return** 1
12: $\quad\quad$ **if** $\text{received}^P_{\text{key}}$ not prefix of $\text{sent}^{\overline{P}}_{\text{key}}$ **then return** 1
13: $\quad$ **else**
14: $\quad\quad$ upd $\leftarrow$ RATCH(P, send)
15: $\quad\quad$ push upd to $\text{in\_queue}^{\overline{P}}$
16: $\quad$ **end if**
17: **end loop**

KIND Security

For all ppt $\mathcal{A}$, $\left|\Pr\left[\mathsf{KIND}^{\mathcal{A}}_{0,C_{\text{clean}}} \to 1\right] - \Pr\left[\mathsf{KIND}^{\mathcal{A}}_{1,C_{\text{clean}}} \to 1\right]\right| = \text{negl}(\lambda)$

Game $\mathsf{KIND}^{\mathcal{A}}_{b,C_{\text{clean}}}$
1: set all variables to $\perp$
2: $\text{Init}(1^\lambda) \xrightarrow{\$} (\text{st}_A, \text{st}_B, z)$
3: $b' \leftarrow \mathcal{A}^{\text{RATCH},\text{EXP}_{\text{st}},\text{EXP}_{\text{key}},\text{TEST}}(z)$
4: **if** $\neg C_{\text{clean}}$ **then** abort
5: **return** b'

exclude trivial attacks

- the EXP oracles can be used for trivial attacks without forgeries
- not easy to identify trivial attacks in the case of forgeries

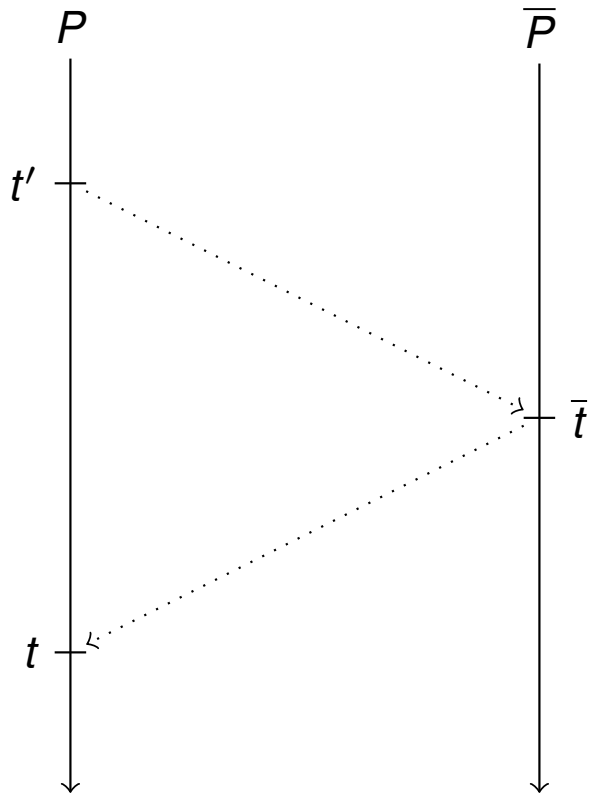
Oracle $\text{TEST}(P)$
1: **if** $t_{\text{test}} \neq \perp$ **then** abort
2: **if** $k_P = \perp$ **then** abort
3: $t_{\text{test}} \leftarrow$ time
4: **if** $b = 1$ **then**
5: **return** k_P
6: **else**
7: **return** random $\{0,1\}^{|k_P|}$
8: **end if**

Oracle $\text{EXP}_{\text{key}}(P)$
1: **return** k_P

Oracle $\text{EXP}_{\text{st}}(P)$
1: **return** st_P

A Few Technical Notions: Matching Status

$$P \text{ in matching status at time } t \iff \exists \overline{t}, t' \begin{cases} t' \leq t \\ \text{received}^{P}_{\text{msg}}(t) = \text{sent}^{\overline{P}}_{\text{msg}}(\overline{t}) \\ \text{received}^{\overline{P}}_{\text{msg}}(\overline{t}) = \text{sent}^{P}_{\text{msg}}(t') \end{cases}$$



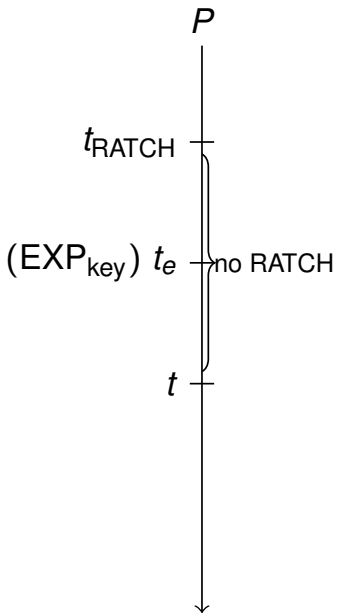
Property

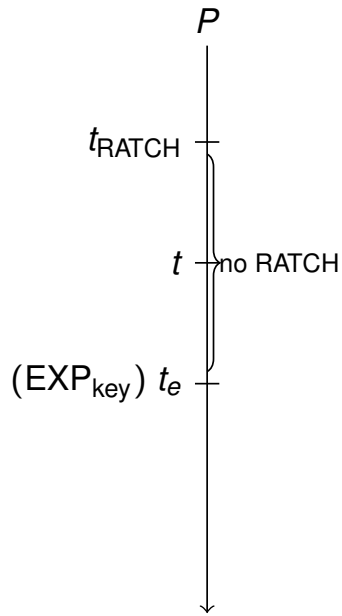
If P in matching status at time t...

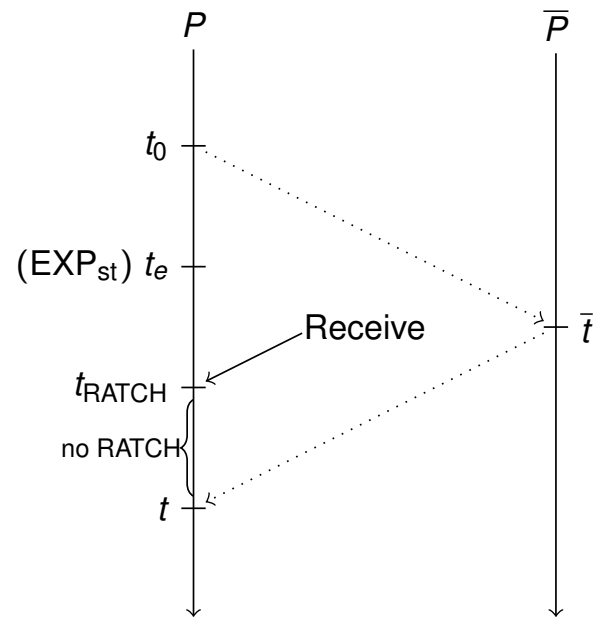
- ...$\overline{P}$ in matching status at time $\overline{t}$
- ...P in matching status at any time $< t$
- ...P and $\overline{P}$ generate the same keys

A Few Technical Notions: Direct Leakage

$k_P(t)$ directly leaks if we are in one of those configurations:





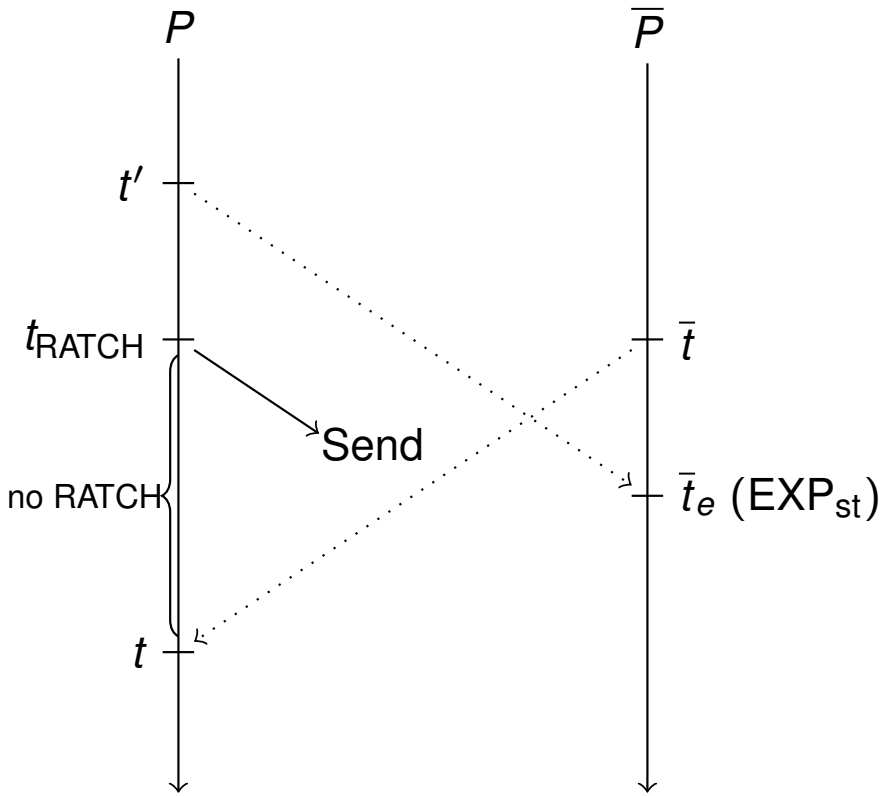


(P in matching status at time t)

A Few Technical Notions: Indirect Leakage

$k_P(t)$ indirectly leaks if P is in matching status at time t and

- either the corresponding $k_{\overline{P}}(\bar{t})$ directly leaks
- or we are in this configuration:



A Few Cleanness Notions

- C_{leak}: $k_{P_{\text{test}}}(t_{\text{test}})$ does leaks neither directly nor indirectly
  mandatory: we must have this clause in C_{clean}
- $C^{P_{\text{test}}}_{\text{trivial forge}}$: P_{test} had no trivial forgery before seeing upd_{test}
- $C^{A,B}_{\text{trivial forge}}$: neither A nor B had a trivial forgery before seeing upd_{test}
- C_{ratchet}: upd_{test} was sent by a participant P, then accepted by $\overline{P}$, then $\overline{P}$ sent some upd, then it was accepted by P

$$
\begin{array}{ll}
(C_{\text{leak}} \wedge C^{P_{\text{test}}}_{\text{trivial forge}})\text{-KIND security} & \leftarrow \text{PR18 and JS18} \\
\Downarrow & \\
(C_{\text{leak}} \wedge C^{A,B}_{\text{trivial forge}})\text{-KIND security} & \leftarrow \text{our protocol} \\
\Downarrow^{(++)} & \\
(C_{\text{leak}} \wedge C_{\text{ratchet}})\text{-KIND security} & \leftarrow \text{we are happy here}
\end{array}
$$

FORGE Security

For all ppt $\mathcal{A}$, $\Pr[\text{FORGE}^{\mathcal{A}} \to 1] = \text{negl}(\lambda)$

Game $\text{FORGE}^{\mathcal{A}}$

1: $\text{Init}(1^{\lambda}) \xrightarrow{\$} (\text{st}_A, \text{st}_B, z)$
2: $(P, \text{upd}) \leftarrow \mathcal{A}^{\text{RATCH},\text{EXP}_{\text{st}},\text{EXP}_{\text{key}}}(z)$
3: **if** there is a participant NOT in a matching status **then return** 0
4: $\text{RATCH}(P, \text{rec}, \text{upd}) \to \text{acc}$
5: **if** $\text{acc} = \text{false}$ **then return** 0
6: **if** P is in a matching status **then return** 0
7: **if** upd is a trivial forgery for P **then return** 0
8: **return** 1

This notion is interesting to have in order to reduce exclusion of forgeries to exclusion of trivial forgeries in KIND security:

$$(C_{\text{leak}} \wedge C^{\star}_{\text{forge}})\text{-KIND security} \implies (C_{\text{leak}} \wedge C^{\star}_{\text{trivial forge}})\text{-KIND security}$$

RECOVER Security

For all ppt $\mathcal{A}$, $\Pr[\text{RECOVER}^{\mathcal{A}} \to 1] = \text{negl}(\lambda)$

Game $\text{RECOVER}^{\mathcal{A}}$

1: $\text{Init}(1^\lambda) \xrightarrow{\$} (\text{st}_A, \text{st}_B, z)$
2: set all lists to $\emptyset$
3: $P \leftarrow \mathcal{A}^{\text{RATCH},\text{EXP}_{\text{st}},\text{EXP}_{\text{key}}}(z)$
4: **if** we can parse as follows **then return** 1

$$\begin{aligned} \text{sent}^{\overline{P}}_{\text{msg}} &= ([\text{seq}_2], \text{upd}, [\text{seq}_3]) \\ &\quad \nvDash \quad \| \\ \text{received}^{P}_{\text{msg}} &= ([\text{seq}_1], \text{upd}) \end{aligned}$$

5: **return** 0

This notion is interesting to have in order to make sure that a round trip communication between honest participants implies no forgery.

$(C_{\text{leak}} \wedge C^{A,B}_{\text{trivial forge}})$-KIND security $\Longrightarrow$ $(C_{\text{leak}} \wedge C_{\text{ratchet}})$-KIND security

Post-Compromise Security Implies PKC

Theorem

Given a correct and weak*-*KIND*-secure unidirectional ARK, we can construct a correct and IND-CPA-secure KEM.*

Game weak-KIND$_b^{\mathcal{A}}$
1: set all variables to $\perp$
2: Init$(1^\lambda) \xrightarrow{\$} (\mathsf{st}_S, \mathsf{st}_R, z)$
3: $b' \leftarrow \mathcal{A}(z)$:
$\mathsf{st} \leftarrow \mathsf{EXP}_{\mathsf{st}}(S)$
$\mathsf{upd} \leftarrow \mathsf{RATCH}(S, \mathsf{send})$
$k \leftarrow \mathsf{TEST}(S)$
$b' \leftarrow \mathcal{A}'(z, \mathsf{st}, \mathsf{upd}, k)$
4: **return** b'

KEM.Gen $\xrightarrow{\$} (\mathsf{sk}, \mathsf{pk})$:
1: Init $\xrightarrow{\$} (\mathsf{st}_S, \mathsf{st}_R, z)$
2: set $\mathsf{pk} = \mathsf{st}_S$, $\mathsf{sk} = \mathsf{st}_R$

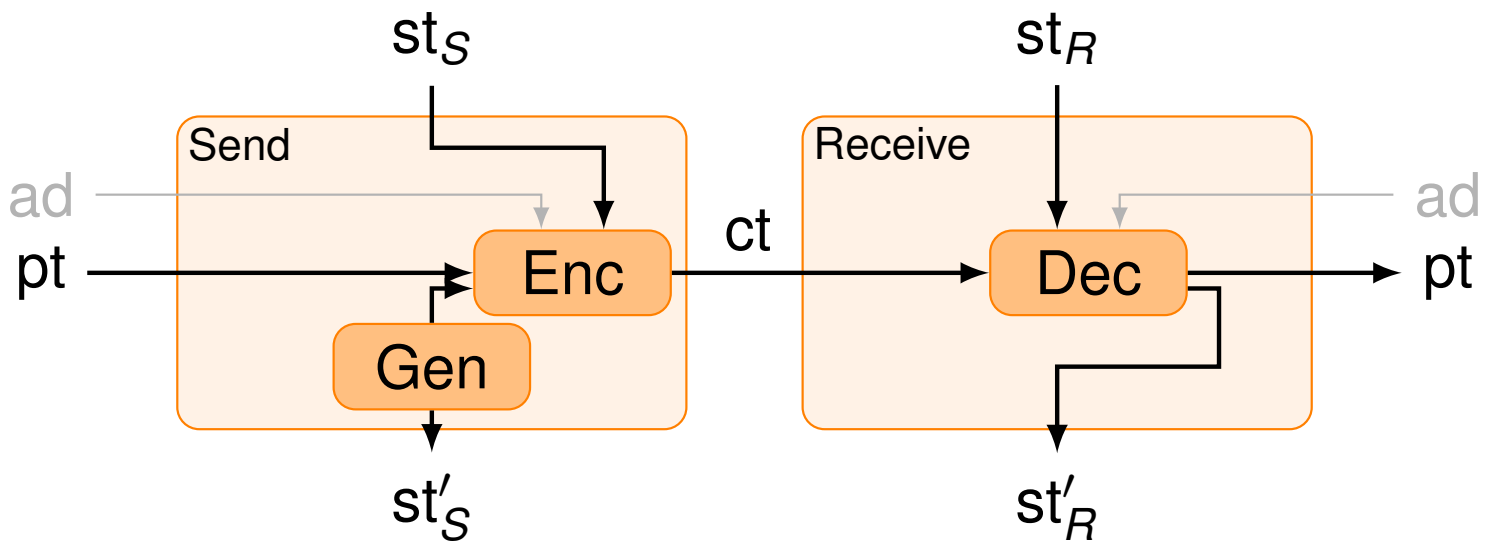
KEM.Enc(pk) $\xrightarrow{\$} (k, \mathsf{ct})$:
3: Send(pk) $\xrightarrow{\$} (., \mathsf{upd}, k)$
4: set $\mathsf{ct} = \mathsf{upd}$

KEM.Dec(sk, ct) $\rightarrow k$:
5: Receive(sk, upd) $\rightarrow (., ., k)$

Our Protocol: uniARCAD

$$\mathsf{SC.Enc}(\overbrace{\mathsf{sk}_S, \mathsf{pk}_R}^{\mathsf{st}_S}, \mathsf{ad}, \mathsf{pt}) = \mathsf{Encrypt}(\mathsf{pk}_R, (\mathsf{pt}, \mathsf{Sign}(\mathsf{sk}_S, (\mathsf{ad}, \mathsf{pt}))))$$

$$\mathsf{SC.Dec}(\underbrace{\mathsf{sk}_R, \mathsf{pk}_S}_{\mathsf{st}_R}, \mathsf{ad}, \mathsf{ct}) = \begin{cases} (\mathsf{pt}, \sigma) \leftarrow \mathsf{Decrypt}(\mathsf{sk}_R, \mathsf{ct}) \\ \mathsf{Verify}(\mathsf{pk}_S, \sigma, (\mathsf{ad}, \mathsf{pt})) \quad ? \quad \mathsf{pt} \quad : \quad \bot \end{cases}$$



uniARCAD.Init(1^λ)
1: $\mathsf{SC.Gen}(1^\lambda) \xrightarrow{\$} (\mathsf{st}_S, \mathsf{st}_R)$
2: **return** $(\mathsf{st}_S, \mathsf{st}_R)$

uniARCAD.Send($\mathsf{st}_S, \mathsf{ad}, \mathsf{pt}$)
1: $\mathsf{SC.Gen}(1^\lambda) \xrightarrow{\$} (\mathsf{st}'_S, \mathsf{st}'_R)$
2: $\mathsf{pt}' \leftarrow (\mathsf{st}'_R, \mathsf{pt})$
3: $\mathsf{ct} \leftarrow \mathsf{SC.Enc}(\mathsf{st}_S, \mathsf{ad}, \mathsf{pt}')$
4: **return** $(\mathsf{st}'_S, \mathsf{ct})$

uniARCAD.Receive($\mathsf{st}_R, \mathsf{ad}, \mathsf{ct}$)
1: $\mathsf{SC.Dec}(\mathsf{st}_R, \mathsf{ad}, \mathsf{ct}) \rightarrow \mathsf{pt}'$
2: **if** $\mathsf{pt}' = \bot$ **then**
3: **return** $(\mathsf{false}, \mathsf{st}_R, \bot)$
4: **end if**
5: parse $\mathsf{pt}' = (\mathsf{st}'_R, \mathsf{pt})$
6: **return** $(\mathsf{true}, \mathsf{st}'_R, \mathsf{pt})$

Our Protocol: BARK (Init)

BARK.Init(1^λ)

1: uniARCAD.Init(1^λ) $\xrightarrow{\$}$ $(\mathsf{st}_A^{\mathsf{send}}, \mathsf{st}_B^{\mathsf{rec}}, z_{A\to B})$
2: uniARCAD.Init(1^λ) $\xrightarrow{\$}$ $(\mathsf{st}_B^{\mathsf{send}}, \mathsf{st}_A^{\mathsf{rec}}, z_{B\to A})$
3: $H.\mathsf{Gen}(1^\lambda) \xrightarrow{\$} \mathsf{hk}$
4: $\mathsf{st}_A \leftarrow (\mathsf{hk}, (\mathsf{st}_A^{\mathsf{send}}), (\mathsf{st}_A^{\mathsf{rec}}), \bot, \bot)$
5: $\mathsf{st}_B \leftarrow (\mathsf{hk}, (\mathsf{st}_B^{\mathsf{send}}), (\mathsf{st}_B^{\mathsf{rec}}), \bot, \bot)$
6: $z \leftarrow (z_{A\to B}, z_{B\to A})$
7: **return** $(\mathsf{st}_A, \mathsf{st}_B, z)$

$$\mathsf{st} = \begin{pmatrix} \langle \text{hash key} \rangle \\ \langle \text{list of send states} \rangle \\ \langle \text{list of receive states} \rangle \\ \langle \text{sent hash} \rangle \\ \langle \text{receive hash} \rangle \end{pmatrix}$$

Onion Encryption

Our Protocol: BARK (Send)

BARK.Send(st_P)

1: parse $\mathsf{st}_P = (\mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v}), \mathsf{Hsent}, \mathsf{Hreceived})$
2: pick k
3: $\mathsf{uniARCAD.Init}(1^\lambda) \xrightarrow{\$} (\mathsf{st}_{S\mathsf{new}}, \mathsf{st}_P^{\mathsf{rec},v+1}, z)$ ▷ append a new receive state to the $\mathsf{st}_P^{\mathsf{rec}}$ list
4: $\mathsf{onion} \leftarrow (\mathsf{st}_{S\mathsf{new}}, k)$ ▷ then, $\mathsf{st}_{S\mathsf{new}}$ is erased to avoid leaking
5: take the smallest i s.t. $\mathsf{st}_P^{\mathsf{send},i} \neq \perp$ ▷ $i = u - n$ if we had n Receive since the last Send
6: **for** $j = u$ down to i **do** ▷ add encryption layers to onion and update $\mathsf{st}_P^{\mathsf{send}}$
7: $\quad \mathsf{uniARCAD.Send}(\mathsf{st}_P^{\mathsf{send},j}, (u - j, \mathsf{Hsent}), \mathsf{onion}) \xrightarrow{\$} (\mathsf{st}_P^{\mathsf{send},j}, \mathsf{onion})$ ▷ update $\mathsf{st}_P^{\mathsf{send},j}$
8: $\quad$ **if** $j < u$ **then** $\mathsf{st}_P^{\mathsf{send},j} \leftarrow \perp$ ▷ flush the send state list: only $\mathsf{st}_P^{\mathsf{send},u}$ remains
9: **end for**
10: $\mathsf{upd} \leftarrow (u - i, \mathsf{Hsent}, \mathsf{onion})$ ▷ the onion has $u - i + 1 = n + 1$ layers
11: $\mathsf{Hsent}' \leftarrow H.\mathsf{Eval}(\mathsf{hk}, \mathsf{upd})$
12: $\mathsf{st}'_P \leftarrow (\mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v+1}), \mathsf{Hsent}', \mathsf{Hreceived})$
13: **return** $(\mathsf{st}'_P, \mathsf{upd})$

- create a new uniARCAD channel for return
- add $\mathsf{st}^{\mathsf{rec}}$ in list of receive states
- concatenate $\mathsf{st}_{S\mathsf{new}}$ to key
- uniARCAD.encrypt with all send states (onion encryption)
- authenticate sent hash and the onion depth

Our Protocol: BARK (Receive)

BARK.Receive(st_P, upd)

1: parse $\mathsf{st}_P = (\mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v}), \mathsf{Hsent}, \mathsf{Hreceived})$
2: parse $\mathsf{upd} = (n, h, \mathsf{onion})$ ▷ the onion has $n+1$ layers
3: **if** $h \neq \mathsf{Hreceived}$ **then return** $(\mathsf{false}, \mathsf{st}_P, \perp)$
4: set i to the smallest index such that $\mathsf{st}_P^{\mathsf{rec},i} \neq \perp$
5: **if** $i + n > v$ **then return** $(\mathsf{false}, \mathsf{st}_P, \perp)$
6: **for** $j = i$ to $i + n$ **do** ▷ peel off onion and compute the next $\mathsf{st}_P^{\mathsf{rec}}$ if accepted
7: $\quad$ uniARCAD.Receive$(\mathsf{st}_P^{\mathsf{rec},j}, (i + n - j, \mathsf{Hreceived}), \mathsf{onion}) \to (\mathsf{acc}, \mathsf{st}'_P{}^{\mathsf{rec},j}, \mathsf{onion})$
8: $\quad$ **if** $\mathsf{acc} = \mathsf{false}$ **then return** $(\mathsf{false}, \mathsf{st}_P, \perp)$
9: **end for**
10: parse $\mathsf{onion} = (\mathsf{st}_P^{\mathsf{send},u+1}, k)$ ▷ a new send state is added in the list
11: **for** $j = i$ to $i + n - 1$ **do** ▷ update $\mathsf{st}_P^{\mathsf{rec}}$ stage 1: clean up
12: $\quad$ $\mathsf{st}_P^{\mathsf{rec},j} \leftarrow \perp$
13: **end for** ▷ n entries of $\mathsf{st}_P^{\mathsf{rec}}$ were erased
14: $\mathsf{st}_P^{\mathsf{rec},i+n} \leftarrow \mathsf{st}'_P{}^{\mathsf{rec},i+n}$ ▷ update $\mathsf{st}_P^{\mathsf{rec}}$ stage 2: update $\mathsf{st}_P^{\mathsf{rec},i+n}$
15: $\mathsf{Hreceived}' \leftarrow H.\mathsf{Eval}(\mathsf{hk}, \mathsf{upd})$
16: $\mathsf{st}'_P \leftarrow (\mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u+1}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v}), \mathsf{Hsent}, \mathsf{Hreceived}')$
17: **return** $(\mathsf{acc}, \mathsf{st}'_P, k)$

- uniARCAD.decrypt with receive states (onion encryption)
- authenticate received hash and the onion depth
- remove all but the last used receive states
- get $\mathsf{st}^{\mathsf{send}}$ and add in list

Example

	Alice			Bob		
	send states	receive states	messages	send states	receive states	
	$\mathrm{st}^{A,S}_{1,0}$	$\mathrm{st}^{A,R}_{1,0}$		$\mathrm{st}^{B,S}_{1,0}$	$\mathrm{st}^{B,R}_{1,0}$	
send k^A_1	$\mathrm{st}^{A,S}_{1,1}$	$\mathrm{st}^{A,R}_{1,0}, \mathrm{st}^{A,R}_{2,0}$	$\rightarrow [\mathrm{st}^{B,S}_{2,0}, k^A_1]_{\mathrm{st}_{1,0}} \rightarrow$	$\mathrm{st}^{B,S}_{1,0}$	$\mathrm{st}^{B,R}_{1,0}$	
send k^A_2	$\mathrm{st}^{A,S}_{1,2}$	$\mathrm{st}^{A,R}_{1,0}, \mathrm{st}^{A,R}_{2,0}, \mathrm{st}^{A,R}_{3,0}$	$\rightarrow [\mathrm{st}^{B,S}_{3,0}, k^A_2]_{\mathrm{st}_{1,1}} \rightarrow$	$\mathrm{st}^{B,S}_{1,0}$	$\mathrm{st}^{B,R}_{1,0}$	
	$\mathrm{st}^{A,S}_{1,2}$	$\mathrm{st}^{A,R}_{1,0}, \mathrm{st}^{A,R}_{2,0}, \mathrm{st}^{A,R}_{3,0}$	$\leftarrow [\mathrm{st}^{A,S}_{2,0}, k^B_1]_{\mathrm{st}_{1,0}} \leftarrow$	$\mathrm{st}^{B,S}_{1,1}$	$\mathrm{st}^{B,R}_{1,0}, \mathrm{st}^{B,R}_{2,0}$	send k^B_1
receive k^B_1	$\mathrm{st}^{A,S}_{1,2}, \mathrm{st}^{A,S}_{2,0}$	$\mathrm{st}^{A,R}_{1,1}, \mathrm{st}^{A,R}_{2,0}, \mathrm{st}^{A,R}_{3,0}$		$\mathrm{st}^{B,S}_{1,1}$	$\mathrm{st}^{B,R}_{1,0}, \mathrm{st}^{B,R}_{2,0}$	
	$\mathrm{st}^{A,S}_{1,2}, \mathrm{st}^{A,S}_{2,0}$	$\mathrm{st}^{A,R}_{1,1}, \mathrm{st}^{A,R}_{2,0}, \mathrm{st}^{A,R}_{3,0}$		$\mathrm{st}^{B,S}_{1,1}, \mathrm{st}^{B,S}_{2,0}$	$\mathrm{st}^{B,R}_{1,1}, \mathrm{st}^{B,R}_{2,0}$	receive k^A_1
	$\mathrm{st}^{A,S}_{1,2}, \mathrm{st}^{A,S}_{2,0}$	$\mathrm{st}^{A,R}_{1,1}, \mathrm{st}^{A,R}_{2,0}, \mathrm{st}^{A,R}_{3,0}$		$\mathrm{st}^{B,S}_{1,1}, \mathrm{st}^{B,S}_{2,0}, \mathrm{st}^{B,S}_{3,0}$	$\mathrm{st}^{B,R}_{1,2}, \mathrm{st}^{B,R}_{2,0}$	receive k^A_2
	$\mathrm{st}^{A,S}_{1,2}, \mathrm{st}^{A,S}_{2,0}$	$\mathrm{st}^{A,R}_{1,1}, \mathrm{st}^{A,R}_{2,0}, \mathrm{st}^{A,R}_{3,0}$	$\leftarrow [\mathrm{st}^{A,S}_{3,0}, k^B_2]_{\mathrm{st}_{1,1},\mathrm{st}_{2,0},\mathrm{st}_{3,0}} \leftarrow$	$\mathrm{st}^{B,S}_{3,1}$	$\mathrm{st}^{B,R}_{1,2}, \mathrm{st}^{B,R}_{2,0}, \mathrm{st}^{B,R}_{3,0}$	send k^B_2
receive k^B_2	$\mathrm{st}^{A,S}_{1,2}, \mathrm{st}^{A,S}_{2,0}, \mathrm{st}^{A,S}_{3,0}$	$\mathrm{st}^{A,R}_{3,1}$		$\mathrm{st}^{B,S}_{3,1}$	$\mathrm{st}^{B,R}_{1,2}, \mathrm{st}^{B,R}_{2,0}, \mathrm{st}^{B,R}_{3,0}$	
send k^A_3	$\mathrm{st}^{A,S}_{3,1}$	$\mathrm{st}^{A,R}_{3,1}, \mathrm{st}^{A,R}_{4,0}$	$\rightarrow [\mathrm{st}^{B,S}_{4,0}, k^A_3]_{\mathrm{st}_{1,2},\mathrm{st}_{2,0},\mathrm{st}_{3,0}} \rightarrow$	$\mathrm{st}^{B,S}_{3,1}$	$\mathrm{st}^{B,R}_{1,2}, \mathrm{st}^{B,R}_{2,0}, \mathrm{st}^{B,R}_{3,0}$	
	$\mathrm{st}^{A,S}_{3,1}$	$\mathrm{st}^{A,R}_{3,1}, \mathrm{st}^{A,R}_{4,0}$		$\mathrm{st}^{B,S}_{3,1}, \mathrm{st}^{B,S}_{4,0}$	$\mathrm{st}^{B,R}_{3,1}$	receive k^A_3

liteBARK: Our Symmetric Protocol

- same as previous protocol with AE instead of SC
- much faster
- no post-compromise security
- still forward security

On-Demand Ratcheting

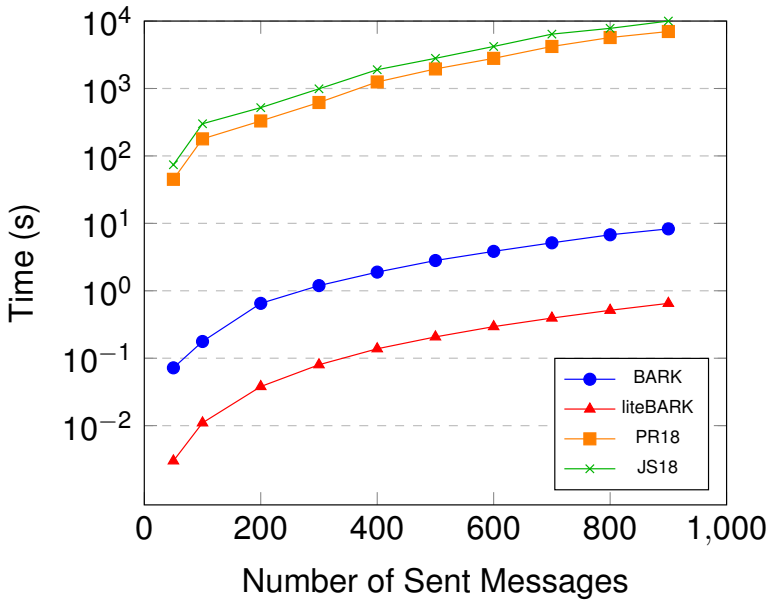
- new interface for Send:

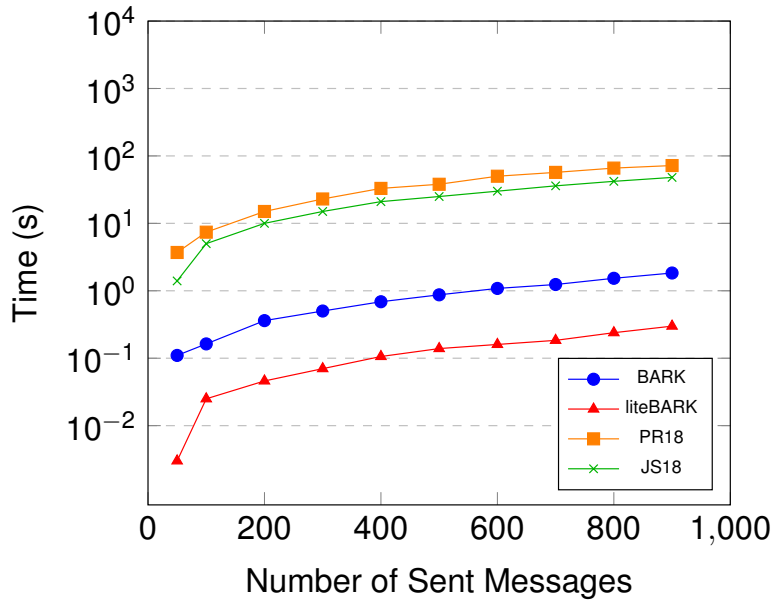
$$\text{Send}(\text{st}, \text{flag})$$

  ratchet iff flag is true
  otherwise, live with symmetric crypto
- use BARK for ratcheting
- every uniARCAD.Init creates new liteBARK states
- sending with flag $=$ false is using liteBARK.Send
- security notion still to come...

Performance

Total amount of time (log scale) to send several messages, in only one direction (left) or in alternating directions (right).





BARK uses ECDSA and ECIES.

liteBARK uses BARK and AES-GCM.

PR18 uses Gentry-Silverberg HIBE and ECDSA.

JS18 uses Gentry-Silverberg HIBE and Bellare-Miner forward-secure signature.

Conclusion

- better understanding on ratcheting security
- ratcheting security can be efficient
- with no HIBE and no random oracle!