A Proper Security Level for Postcompromise Secure Messaging

F Betül Durak and Serge Vaudenay

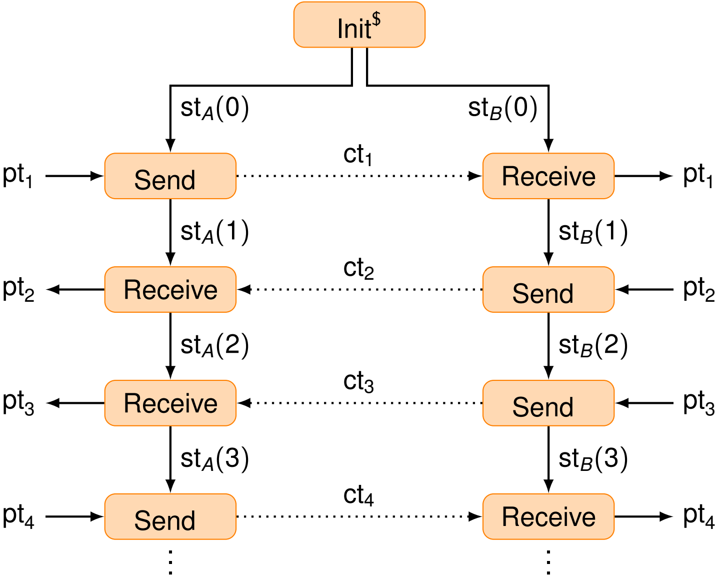
New logo!

1. Ratcheting
2. BARK
3. ARCAD
4. Comparison of Protocols

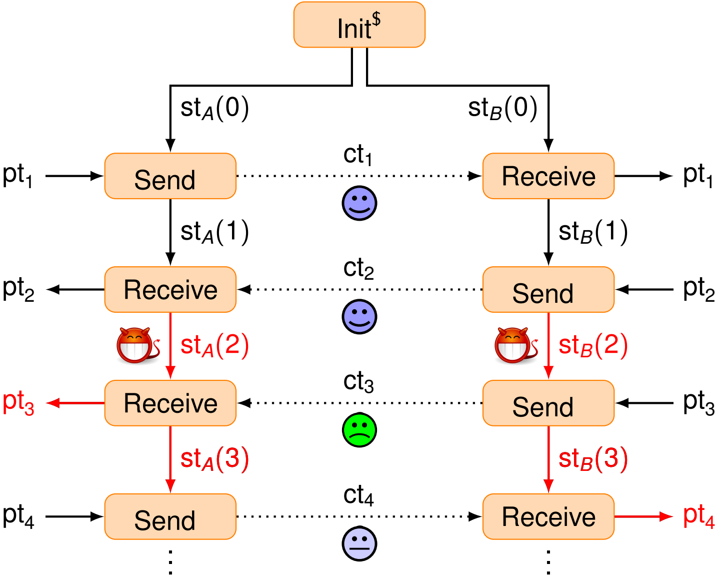
1. Ratcheting
2. BARK
3. ARCAD
4. Comparison of Protocols

End-To-End Secure IM

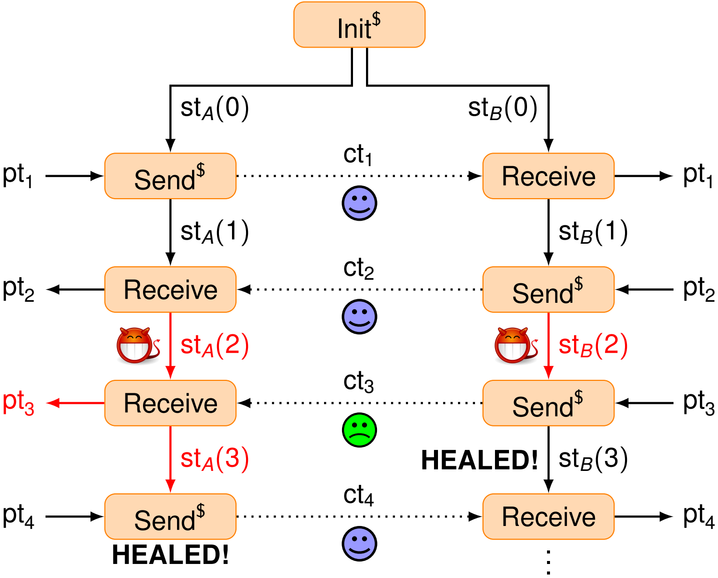
Secure Bidirectional Communication



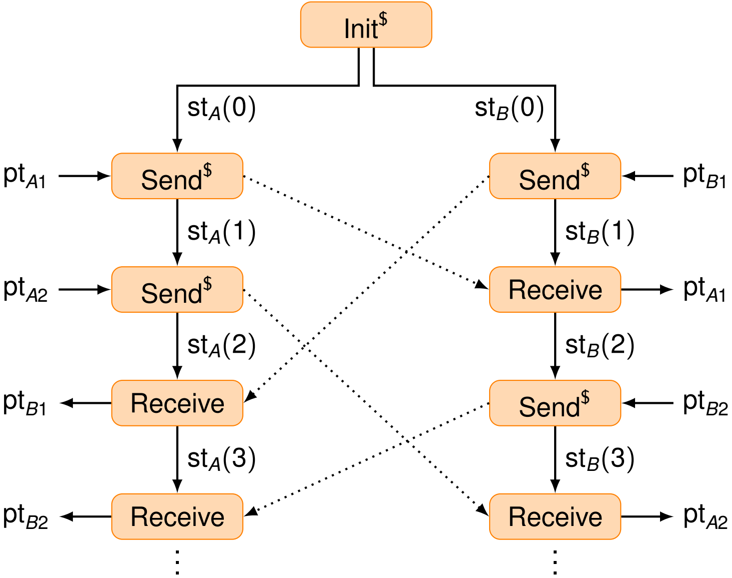
Aim: Forward Secrecy



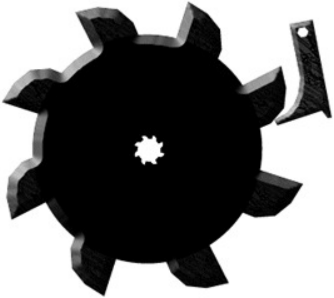
Aim: + Post-Compromise Security



By the Way: Asynchronous + Random Role



Ratchet



state update

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

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

EUROCRYPT 2019

with immediate decryption

Alwen, Coretti, Dodis
The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol

Jost, Maurer, Mularczyk
Efficient Ratcheting: Almost-Optimal Guarantees for Secure Messaging

near-optimal security but better complexity — still high

Our Results

Durak, Vaudenay
Bidirectional Asynchronous Ratcheted Key Agreement with Linear Complexity
Eprint 2018/889

Caforio, Durak, Vaudenay
On-Demand Ratcheting with Security Awareness
Soon on Eprint

1. Ratcheting
2. **BARK**
3. ARCAD
4. Comparison of Protocols

BARK

Bidirectional Asynchronous Ratcheted Key Agreement

Interface

- $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$ (the common public parameters)
- $\mathsf{Gen}(\mathsf{pp}) \xrightarrow{\$} (\mathsf{sk}, \mathsf{pk})$ (key pair of a participant)
- $\mathsf{Init}(\mathsf{pp}, \mathsf{sk}_P, \mathsf{pk}_{\overline{P}}, P) \rightarrow \mathsf{st}_P$ (initial state)
- $\mathsf{Send}(\mathsf{st}_P) \xrightarrow{\$} (\mathsf{st}'_P, \mathsf{ct}, k)$ (like KEM.Enc)
- $\mathsf{Receive}(\mathsf{st}_P, \mathsf{ct}) \rightarrow (\mathsf{acc}, \mathsf{st}'_P, k)$ (like KEM.Dec)

InitAll(pp):
1: $\mathsf{Gen}(\mathsf{pp}) \rightarrow (\mathsf{sk}_A, \mathsf{pk}_A)$
2: $\mathsf{Gen}(\mathsf{pp}) \rightarrow (\mathsf{sk}_B, \mathsf{pk}_B)$
3: $\mathsf{st}_A \leftarrow \mathsf{Init}(\mathsf{pp}, \mathsf{sk}_A, \mathsf{pk}_B, 0)$
4: $\mathsf{st}_B \leftarrow \mathsf{Init}(\mathsf{pp}, \mathsf{sk}_B, \mathsf{pk}_A, 1)$
5: $z \leftarrow (\mathsf{pp}, \mathsf{pk}_A, \mathsf{pk}_B)$
6: **return** $(\mathsf{st}_A, \mathsf{st}_B, z)$

we must specify what to give to the adversary

Correctness

of form (P, send) or (P, rec)

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

Oracle $\text{RATCH}(P, \text{send})$

$(\text{st}_P, \text{ct}_P, k_P) \leftarrow \text{Send}(\text{st}_P)$
append k_P to $\text{sent}^P_{\text{key}}$
return ct_P

Oracle $\text{RATCH}(P, \text{rec}, \text{ct})$

$(\text{acc}, \text{st}'_P, k'_P) \leftarrow \text{Receive}(\text{st}_P, \text{ct})$
if acc **then**
    $\text{st}_P \leftarrow \text{st}'_P$
    $k_P \leftarrow k'_P$
    append k_P to $\text{received}^P_{\text{key}}$
end if
return acc

Game Correctness(sched)

Setup $\xrightarrow{\$}$ pp
Initall(pp) $\xrightarrow{\$}$ $(\text{st}_A, \text{st}_B, z)$
initialize two FIFO incoming_P, $P \in \{A, B\}$
$i \leftarrow 0$
loop
    $i \leftarrow i + 1$
    $(P, \text{role}) \leftarrow \text{sched}_i$
    if role = send **then**
        $\text{ct} \leftarrow \text{RATCH}(P, \text{send})$
        push ct to $\text{incoming}_{\overline{P}}$
    else
        if incoming_P is empty **then** **return** 0
        pull ct from incoming_P
        $\text{acc} \leftarrow \text{RATCH}(P, \text{rec}, \text{ct})$
        if acc = false **then** **return** 1
    end if
    if $\text{received}^A_{\text{key}}$ not prefix of $\text{sent}^B_{\text{key}}$ **then** **return** 1
    if $\text{received}^B_{\text{key}}$ not prefix of $\text{sent}^A_{\text{key}}$ **then** **return** 1
end loop

KIND Security

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

Game $\mathsf{KIND}^{\mathcal{A}}_{\color{red}{b},C_{\text{clean}}}$

```
Setup →$ pp
Initall(pp) →$ (st_A, st_B, z)
b' ← A^{RATCH, EXP_st, EXP_key, TEST}(z)
if ¬C_clean then return ⊥
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 $\mathsf{TEST}(P)$

```
if b = 1 then
    return k_P
else
    return random {0,1}^{|k_P|}
end if
```

Oracle $\mathsf{EXP}_{\text{key}}(P)$

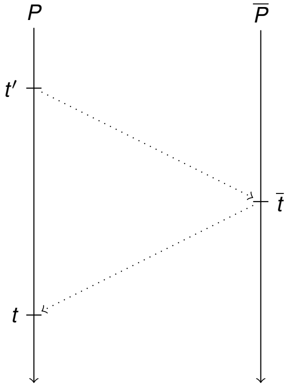
```
return k_P
```

Oracle $\mathsf{EXP}_{\text{st}}(P)$

```
return st_P
```

A Few Technical Notions: Matching Status

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



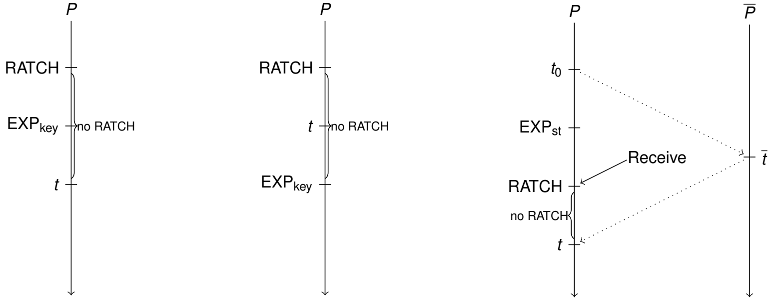
Property

If P in matching status at time t...

- ...$\overline{P}$ in matching status at time $\bar{t}$
- ...P in matching status before
- ...$k_P(t) = k_{\overline{P}}(\bar{t})$

A Few Technical Notions: Direct Leakage

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

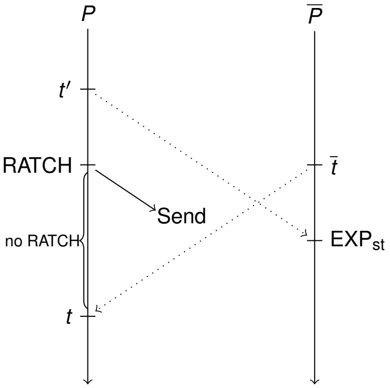


(P in matching status at time t)

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}: the tested $k_{P_{\text{test}}}$ 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 TEST
- $C^{A,B}_{\text{trivial forge}}$: neither A nor B had a trivial forgery before seeing the ct making the tested $k_{P_{\text{test}}}$

$$
\begin{array}{ll}
(C_{\text{leak}} \wedge C^{P_{\text{test}}}_{\text{trivial forge}})\text{-KIND security} & \leftarrow \text{PR18 and JS18 (optimal)} \\
\Downarrow & \\
(C_{\text{leak}} \wedge C^{A,B}_{\text{trivial forge}})\text{-KIND security} & \leftarrow \text{BARK (sub-optimal)}
\end{array}
$$

Why Optimal Security?

- seems to somehow imply HIBE...
- how would P_{test} know he accepted no forgery?
- by making sure that he can still communicate with $\overline{P}_{\text{test}}$
- $\Longrightarrow$ happy with
  C_{ratchet}: the ct making the tested $k_{P_{\text{test}}}$ initiated a round trip
  $P \xrightarrow{\text{ct}} \overline{P} \xrightarrow{\text{ct}'} P$

$$(C_{\text{leak}} \wedge C^{P_{\text{test}}}_{\text{trivial forge}})\text{-KIND security} \quad \leftarrow \text{PR18 and JS18 (optimal)}$$

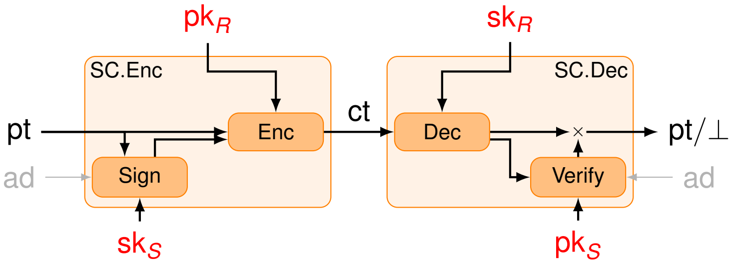
$$\Downarrow$$

$$(C_{\text{leak}} \wedge C^{A,B}_{\text{trivial forge}})\text{-KIND security} \quad \leftarrow \text{BARK (sub-optimal)}$$

$$\Downarrow^{(++)}$$

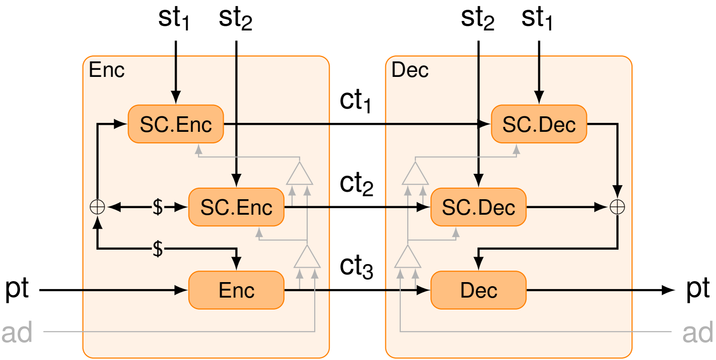
$$(C_{\text{leak}} \wedge C_{\text{ratchet}})\text{-KIND security} \quad \leftarrow \text{we are happy here}$$

A Naive Signcryption

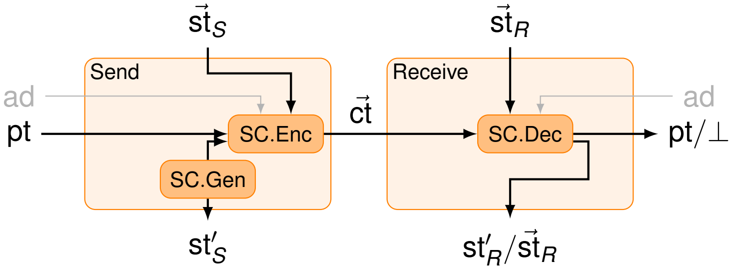


- encrypt and authenticate pt
- can authenticate ad at the same time
- sender state $\mathsf{st}_S = (\mathsf{sk}_S, \mathsf{pk}_R)$
- receiver state $\mathsf{st}_R = (\mathsf{sk}_R, \mathsf{pk}_S)$

Signcryption → Multiple-Key Signcryption (Onion)

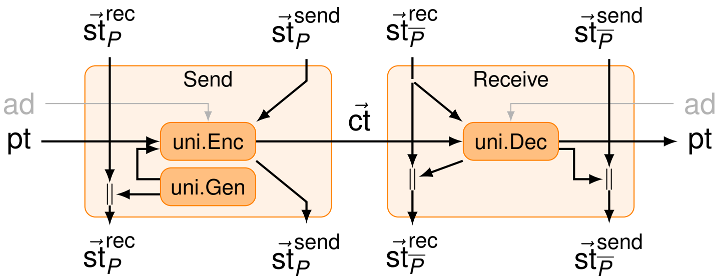


M-Key Signcryption → Unidirectional Ratchet



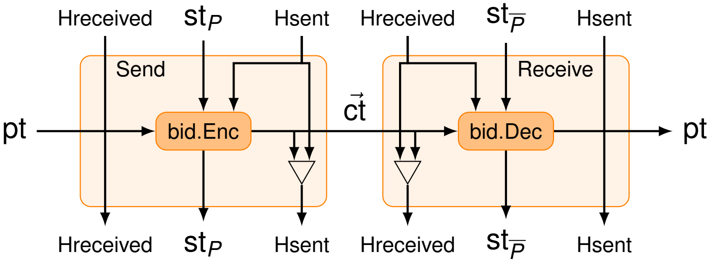
- generate the next send state while sending
- transmit the next receive state while sending
- flush all accumulated states

Unidirectional Ratchet → Bidirectional Ratchet



- generate a state for *replying* at sending
- accumulate receive states at sending

Bidirectional Ratchet $\rightarrow$ BARK



- authenticate the chain of sent messages while sending

Our Protocol: BARK (Setup, Gen, Init)

BARK.Setup

1: $H.\mathsf{Gen}(1^\lambda) \xrightarrow{\$} \mathsf{hk}$
2: **return** hk

BARK.Gen(hk)

1: $\mathsf{SC.Gen}_S \xrightarrow{\$} (\mathsf{sk}_S, \mathsf{pk}_S)$
2: $\mathsf{SC.Gen}_R \xrightarrow{\$} (\mathsf{sk}_R, \mathsf{pk}_R)$
3: $\mathsf{sk} \leftarrow (\mathsf{sk}_S, \mathsf{sk}_R)$
4: $\mathsf{pk} \leftarrow (\mathsf{pk}_S, \mathsf{pk}_R)$
5: **return** (sk, pk)

BARK.Init$(\mathsf{hk}, \mathsf{sk}_P, \mathsf{pk}_{\overline{P}}, P)$

1: parse $\mathsf{sk}_P = (\mathsf{sk}_S, \mathsf{sk}_R)$
2: parse $\mathsf{pk}_{\overline{P}} = (\mathsf{pk}_S, \mathsf{pk}_R)$
3: $\mathsf{st}_P^{\mathsf{send}} \leftarrow (\mathsf{sk}_S, \mathsf{pk}_R)$
4: $\mathsf{st}_P^{\mathsf{rec}} \leftarrow (\mathsf{sk}_R, \mathsf{pk}_S)$
5: $\mathsf{st}_P \leftarrow (\mathsf{hk}, (\mathsf{st}_A^{\mathsf{send}}), (\mathsf{st}_A^{\mathsf{rec}}), \perp, \perp)$
6: **return** st_P

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

Our Protocol: BARK (Send)

$\mathsf{BARK.Send}(\mathsf{st}_P)$

- parse $\mathsf{st}_P = (\mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v}), \mathsf{Hsent}, \mathsf{Hreceived})$
- pick k
- $\mathsf{onion.Init}(1^\lambda) \xrightarrow{\$} (\mathsf{st}_{S\mathsf{new}}, \mathsf{st}_P^{\mathsf{rec},v+1})$ ▷ append a new receive state to the $\mathsf{st}_P^{\mathsf{rec}}$ list
- $\mathsf{pt} \leftarrow (\mathsf{st}_{S\mathsf{new}}, k)$ ▷ then, $\mathsf{st}_{S\mathsf{new}}$ is erased to avoid leaking
- take the smallest i s.t. $\mathsf{st}_P^{\mathsf{send},i} \neq \bot$ ▷ $i = u - n$ if we had n Receive since the last Send
- $\mathsf{onion.Send}(\mathsf{hk}, \mathsf{st}_P^{\mathsf{send},i}, \ldots, \mathsf{st}_P^{\mathsf{send},u}, \mathsf{Hsent}, \mathsf{pt}) \xrightarrow{\$} (\mathsf{st}_P^{\mathsf{send},u}, \mathsf{ct})$ ▷ update $\mathsf{st}_P^{\mathsf{send},u}$
- $\mathsf{st}_P^{\mathsf{send},i}, \ldots, \mathsf{st}_P^{\mathsf{send},u-1} \leftarrow \bot$ ▷ flush the send state list: only $\mathsf{st}_P^{\mathsf{send},u}$ remains
- $\mathsf{ct} \leftarrow (\mathsf{Hsent}, \mathsf{ct})$ ▷ the onion has $u - i + 1 = n + 1$ layers
- $\mathsf{Hsent}' \leftarrow H.\mathsf{Eval}(\mathsf{hk}, \mathsf{ct})$
- $\mathsf{st}'_P \leftarrow (\mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v+1}), \mathsf{Hsent}', \mathsf{Hreceived})$
- **return** $(\mathsf{st}'_P, \mathsf{ct})$

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

Our Protocol: BARK (Receive)

BARK.Receive($\mathsf{st}_P, \mathsf{ct}$)

parse $\mathsf{st}_P = (\mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v}), \mathsf{Hsent}, \mathsf{Hreceived})$
parse $\mathsf{ct} = (h, \mathsf{ct})$ ▷ the onion has $n+1$ layers
set $n+1$ to the number of components in ct
if $h \neq \mathsf{Hreceived}$ **then return** $(\mathsf{false}, \mathsf{st}_P, \perp)$
set i to the smallest index such that $\mathsf{st}_P^{\mathsf{rec},i} \neq \perp$
if $i + n > v$ **then return** $(\mathsf{false}, \mathsf{st}_P, \perp)$
onion.Receive($\mathsf{hk}, \mathsf{st}_P^{\mathsf{rec},i}, \ldots, \mathsf{st}_P^{\mathsf{rec},i+n-1}, \mathsf{Hreceived}, \mathsf{ct}) \rightarrow (\mathsf{acc}, {\mathsf{st}'_P}^{\mathsf{rec},i+n-1}, \mathsf{pt})$
if $\mathsf{acc} = \mathsf{false}$ **then return** $(\mathsf{false}, \mathsf{st}_P, \perp)$
parse $\mathsf{pt} = (\mathsf{st}_P^{\mathsf{send},u+1}, k)$ ▷ a new send state is added in the list
$\mathsf{st}_P^{\mathsf{send},i}, \ldots, \mathsf{st}_P^{\mathsf{send},i+n-2} \leftarrow \perp$ ▷ n entries of $\mathsf{st}_P^{\mathsf{rec}}$ were erased
$\mathsf{st}_P^{\mathsf{rec},i+n-1} \leftarrow {\mathsf{st}'_P}^{\mathsf{rec},i+n-1}$ ▷ update $\mathsf{st}_P^{\mathsf{rec}}$ stage 2: update $\mathsf{st}_P^{\mathsf{rec},i+n}$
$\mathsf{Hreceived}' \leftarrow H.\mathsf{Eval}(\mathsf{hk}, \mathsf{ct})$
$\mathsf{st}'_P \leftarrow (\mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u+1}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v}), \mathsf{Hsent}, \mathsf{Hreceived}')$
return $(\mathsf{acc}, \mathsf{st}'_P, k)$

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

FORGE Security

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

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

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

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

$$(C_{\text{leak}} \wedge C^{\star}_{\text{forge}})\text{-KIND security} \overset{(+\text{FORGE})}{\Longleftarrow\!\!\Longrightarrow} (C_{\text{leak}} \wedge C^{\star}_{\text{trivial forge}})\text{-KIND security}$$

RECOVER Security

For all ppt $\mathcal{A}$, $\Pr[\mathsf{RECOVER}^{\mathcal{A}} \to 1] = \mathsf{negl}$

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

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

$$\begin{array}{rcl} \mathsf{sent}_{\mathsf{msg}}^{\overline{P}} & = & ([\mathsf{seq}_2], \mathsf{ct}, [\mathsf{seq}_3]) \\ & & \quad \not\Vdash \quad \ \| \\ \mathsf{received}_{\mathsf{msg}}^{P} & = & ([\mathsf{seq}_1], \mathsf{ct}) \end{array}$$

6: **return** 0

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

$$(C_{\mathsf{leak}} \wedge C_{\mathsf{trivial\ forge}}^{A,B})\text{-KIND security} \overset{(+\mathsf{RECOVER})}{\Longrightarrow} (C_{\mathsf{leak}} \wedge C_{\mathsf{ratchet}})\text{-KIND security}$$

Security of BARK

Theorem

If

- *H is collision-resistant,*
- *Sign is EF-OTCPA-secure,*
- *PKC is IND-CCA-secure,*
- *Sym is IND-OTCCA-secure,*

then BARK is

- *RECOVER-secure,*
- *FORGE-secure, and*
- *KIND-secure for cleanness* $C_{\text{leak}} \wedge C^{A,B}_{\text{trivial forge}}$.

1 Ratcheting

2 BARK

3 **ARCAD**

4 Comparison of Protocols

ARCAD

Asynchronous Ratcheted Communication with Additional Data

- new interface for Send:

$$\mathsf{Send}(\mathsf{st}, \mathsf{ad}, \mathsf{pt}) \to \mathsf{st}', \mathsf{ct}$$

  encrypt pt and authenticate ad at the same time

- new interface for Receive:

$$\mathsf{Receive}(\mathsf{st}, \mathsf{ad}, \mathsf{ct}) \to \mathsf{acc}, \mathsf{st}', \mathsf{pt}$$

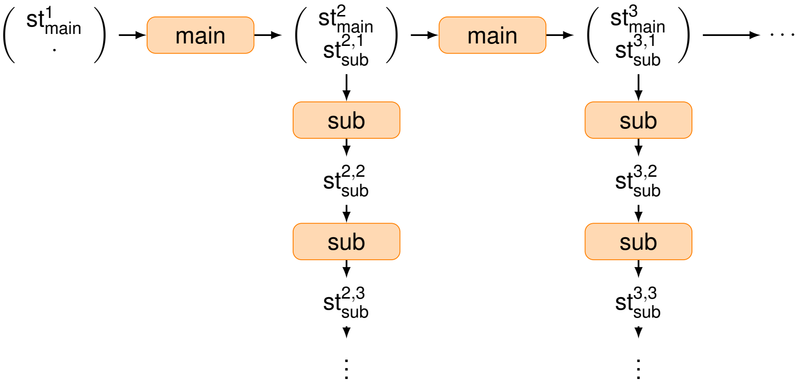
liteARCAD: Our Symmetric Protocol

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

On-Demand Ratcheting

- use a flag in ad denoted by ad.flag
  - ad.flag $=$ true: ratchet
  - ad.flag $=$ false: live with symmetric crypto
- hybrid security notion...
  - adapt BARK as ARCAD_{DV} for ratchet
  - use liteARCAD for symmetric crypto

Hybrid Ratcheting



Hybrid Ratcheting: Results

- combining $\text{ARCAD}_{\text{DV}} + \text{liteARCAD}$, we obtain the best performances if we scarsely ratchet
- privacy is preserved (with hybrid cleanness...)
- unforgeability degrades a bit
- a final protocol transformation restores unforgeability

Security Awareness

- **r-RECOVER security**: cannot *receive* any genuine message after receiving a forgery
- **s-RECOVER security**: cannot *send* any genuine message after receiving a forgery
- **acknowledgement extractor**: each message carries and ACK of received messages
- **cleanness extractor**: can figure out which message remains private from the history of queries

$\rightarrow$ achieved with hybrid ARCAD_{DV} $+$ liteARCAD

1 Ratcheting

2 BARK

3 ARCAD

4 **Comparison of Protocols**

Implementations

ARCAD_{DV} uses ECDSA and ECIES.

liteARCAD uses AES-GCM.

PR18 uses Gentry-Silverberg HIBE and ECDSA.

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

ACD19 uses ECDH and AES-GCM

JMM19 uses ECDSA and ECIES

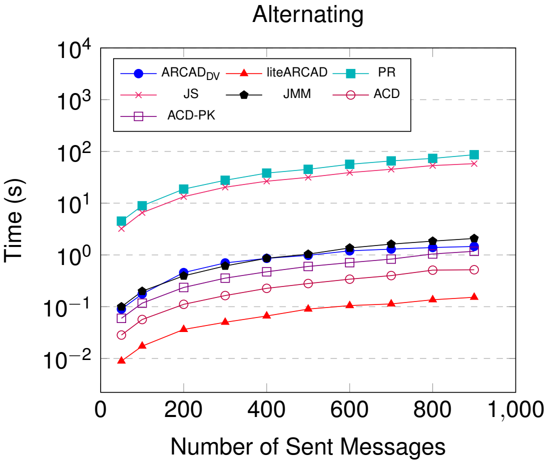
Acknowledgement: implementations by Andrea Caforio

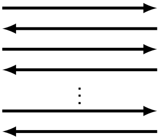
`https://github.com/qantik/ratcheted`

Performance

Runtime

Total amount of time (log scale) to send *n* messages in alternating directions

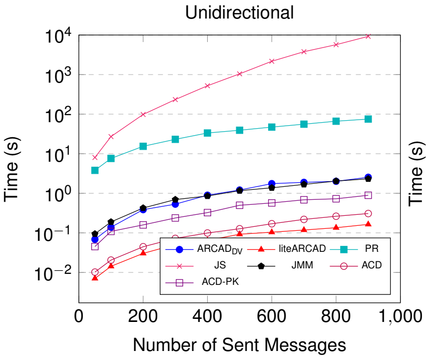


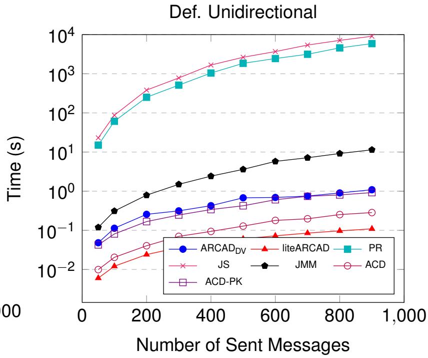


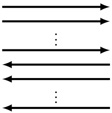
Performance

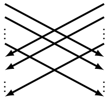
Runtime

Total amount of time (log scale) to send *n* messages





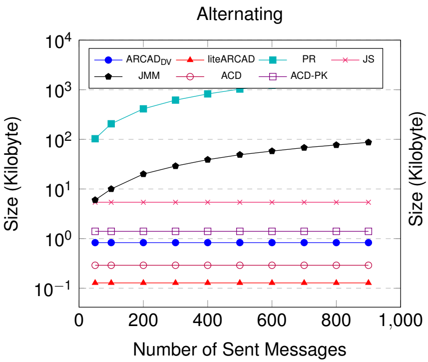


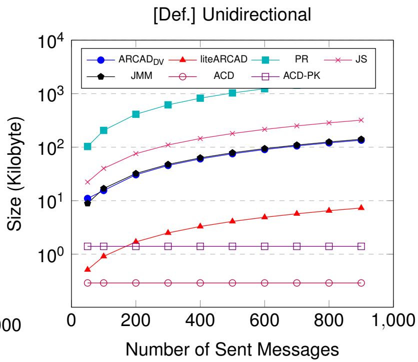


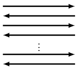
Performance

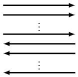
State Size

Maximal state size (log scale) to send *n* messages









Comparison

ARCAD_{DV} + liteARCAD

	PR18	JS18	BARK	JMM19	ACD19-PK	ARCAD
Security	optimal	optimal	sub-optimal	near-optimal	id-optimal	pragmatic
Complexity	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Corrupt coins resilience	no	pre-send $\Rightarrow$ exposure	no	post-send	chosen coins $\Rightarrow$ exposure	no
Plain model	no	no	yes	no	yes	yes
PKC or less	no	no	yes	yes	yes	yes
Immediate decryption	no	no	no	no	yes	no
r-RECOVER security	no	yes	yes	no	no	yes
s-RECOVER security	no	yes	no	no	no	yes
ack. extractor	yes	yes	yes	yes	no	yes
cleanness extractor	yes	yes	yes	yes	yes	yes

- Security: optimal > near-optimal > sub-optimal > pragmatic > id-optimal
- Complexity to send *n* messages in total
- Plain model: some need random oracles
- PKC or less: some need HIBE

Conclusion

- better understanding on ratcheting security
- ratcheting security can be efficient
- new notions: on-demand ratcheting, security awareness