Automatically Invalidating Game-Based Security Definitions

SKECH

June 24

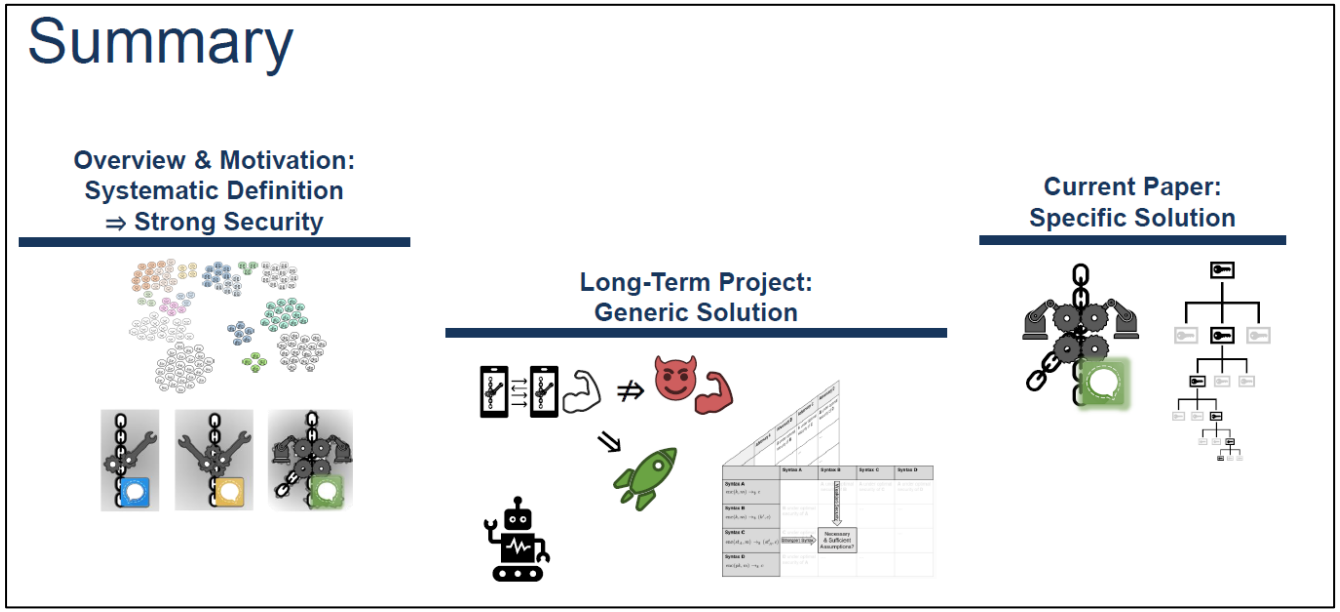
Real-World Cryptography Group
FAU Erlangen-Nürnberg, Germany

Paul Rösler

Disclaimer: Semi-**Confidential**!

- You're *Early Reviewer 3*

- Workshop
- Discussion
- Ideas
- Work in progress:
  - Extensions
  - Part of larger project



Setting: Game-Based Security Definitions

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

$s_A \leftarrow 0;\ r_B \leftarrow 0$
$s_B \leftarrow 0;\ r_A \leftarrow 0$
$e_A \leftarrow 0;\ \mathrm{EP}_A[\cdot] \leftarrow \bot$
$E_B^{\vdash} \leftarrow 0;\ E_B^{\dashv} \leftarrow 0$
$adc_A[\cdot] \leftarrow \bot;\ is_B \leftarrow \mathtt{T}$
$adc_B[\cdot] \leftarrow \bot;\ is_A \leftarrow \mathtt{T}$
$key_A[\cdot] \leftarrow \bot;\ key_B[\cdot] \leftarrow \bot$
$\mathrm{XP}_A \leftarrow \emptyset;\ \mathrm{XP}_B \leftarrow \emptyset$
$\mathrm{TR}_A \leftarrow \emptyset;\ \mathrm{TR}_B \leftarrow \emptyset$
$\mathrm{CH}_A \leftarrow \emptyset;\ \mathrm{CH}_B \leftarrow \emptyset$
$(S_A, S_B) \leftarrow_{\$} \mathrm{init}$
$b' \leftarrow_{\$} \mathcal{A}$
Require $\mathrm{TR}_A \cap \mathrm{CH}_A = \emptyset$
Require $\mathrm{TR}_B \cap \mathrm{CH}_B = \emptyset$
Stop with b'

Oracle $\mathrm{SndA}(ad)$

Require $S_A \neq \bot$
$(S_A, k, c) \leftarrow_{\$} \mathrm{snd}_A(S_A, ad)$
If is_A:
  $adc_A[s_A] \leftarrow (ad, c)$
  $\mathrm{EP}_A[s_A] \leftarrow e_A$
$key_A[e_A, s_A] \leftarrow k$
$s_A \leftarrow s_A + 1$
Return c

Oracle $\mathrm{RcvA}(ad, c)$

Require $S_A \neq \bot$
If $is_A \wedge adc_B[r_A] \neq (ad, c)$:
  $is_A \leftarrow \mathtt{F}$
  If $r_A \in \mathrm{XP}_B$:
    $\mathrm{TR}_A \xleftarrow{\cup} \mathbb{N} \times [s_A, \ldots]$
If is_A: $e_A \leftarrow e_A + 1$
$S_A \leftarrow \mathrm{rcv}_A(S_A, ad, c)$
If $S_A = \bot$: Return $\bot$
$r_A \leftarrow r_A + 1$
Return

Oracle $\mathrm{RcvB}(ad, c)$

Require $S_B \neq \bot$
If $is_B \wedge adc_A[r_B] \neq (ad, c)$:
  $is_B \leftarrow \mathtt{F}$
  If $r_B \in \mathrm{XP}_A$:
    $\mathrm{TR}_B \xleftarrow{\cup} \mathbb{N} \times [r_B, \ldots]$
If is_B: $E_B^{\vdash} \leftarrow \mathrm{EP}_A[r_B]$
$(S_B, k) \leftarrow \mathrm{rcv}_B(S_B, ad, c)$
If $S_B = \bot$: Return $\bot$
If is_B: $k \leftarrow \diamond$
$key_B[E_B^{\vdash}, r_B] \leftarrow k$
$r_B \leftarrow r_B + 1$
Return

Oracle $\mathrm{SndB}(ad)$

Require $S_B \neq \bot$
$(S_B, c) \leftarrow_{\$} \mathrm{snd}_B(S_B, ad)$
If is_B:
  $adc_B[s_B] \leftarrow (ad, c)$
  $E_B^{\dashv} \leftarrow E_B^{\dashv} + 1$
$s_B \leftarrow s_B + 1$
Return c

Oracle ExposeA

If is_A: $\mathrm{XP}_A \xleftarrow{\cup} \{s_A\}$
Return S_A

Oracle ExposeB

$\mathrm{TR}_B \xleftarrow{\cup} [E_B^{\vdash} .. E_B^{\dashv}] \times [r_B, \ldots]$
If is_B:
  $\mathrm{XP}_B \xleftarrow{\cup} \{s_B\}$
  $\mathrm{TR}_A \xleftarrow{\cup} [E_B^{\vdash} .. E_B^{\dashv}] \times [r_B, \ldots]$
Return S_B

Oracle $\mathrm{Reveal}(u, i)$
as in URKE (Fig. 5)

Oracle $\mathrm{Challenge}(u, i)$
as in URKE (Fig. 5)

SoK: Game-based Security Models for Group Key Exchange

Bertram Poettering[1], Paul Rösler[2], Jörg Schwenk[3], and Douglas Stebila[4]

[1] IBM Research – Zurich, Rüschlikon, Switzerland
poe@zurich.ibm.com
[2] TU Darmstadt, Darmstadt, Germany
paul.roesler@tu-darmstadt.de
[3] Ruhr University Bochum, Bochum, Germany
joerg.schwenk@rub.de
[4] University of Waterloo, Waterloo, Canada
dstebila@uwaterloo.ca

Game $\mathrm{ANON}^b_{\mathsf{RKE}}(\mathcal{A})$

$\mathsf{U\text{-}ANON}_b \leftarrow \mathsf{U\text{-}ANON}^b_{\mathsf{RKE}}$
For $d \in \{0, 1\}$:
  $(s_d, r_d) \leftarrow (0, 0)$
  $\mathrm{imp}_d \leftarrow \mathtt{fal}$
$(\boldsymbol{stR}, \boldsymbol{cstR}, \boldsymbol{cad}) \leftarrow ([\cdot], [\cdot], [\cdot])$
$\boldsymbol{stR}[0] \leftarrow \mathsf{U\text{-}ANON}_b.\mathrm{stR}$
$(\boldsymbol{c}, \boldsymbol{cc}, \boldsymbol{rcvd}) \leftarrow (\emptyset, \emptyset, \emptyset)$
$(\boldsymbol{xS}, \boldsymbol{cxS}) \leftarrow (\emptyset, [\cdot])$
$(\mathrm{xS}, \mathrm{cxS}, \mathrm{xR}, \mathrm{cxR}) \leftarrow (\mathtt{fal}, \mathtt{fal}, \mathtt{fal}, \mathtt{fal})$
$b' \xleftarrow{\$} \mathcal{A}$
Stop with b'

Oracle RR

$\mathsf{U\text{-}ANON}_b.\mathrm{RR}$
$(\mathrm{xS}, \mathrm{cxS}) \leftarrow (\mathtt{fal}, \mathtt{fal})$
Return

Oracle Expose_S

▷ If $\mathrm{cxS} = \mathtt{tru}$: Require $(s_0, \_) \notin \boldsymbol{cxS}$
▷ If $\mathrm{xS} = \mathtt{tru} \wedge (s_0, s_1) \notin \boldsymbol{xS}$:
▷ Require $(\_, s_1) \notin \boldsymbol{xS}$
◇ If $\mathrm{imp}_0 = \mathrm{imp}_1 = \mathtt{fal}$:
◇ Require $\mathrm{cxR} = \mathtt{fal}$

$\mathrm{stS} \leftarrow \mathsf{U\text{-}ANON}_b.\mathrm{Expose}_S$
$\boldsymbol{xS} \xleftarrow{\cup} \{(s_0, s_1)\}$
$\mathrm{xS} \leftarrow \mathtt{tru}$
Return stS

Oracle Expose_R

i Require unique $= \mathtt{tru}$
◁ Require $\mathrm{cxR} = \mathtt{fal}$
◇ Require $\mathrm{imp}_0 = \mathrm{imp}_1$
If $\mathrm{imp}_0 = \mathrm{imp}_1 = \mathtt{fal}$:
⊕ For all $\hat{s} \in \boldsymbol{cc}$ require $\hat{s} \leq r_0$
◇ Require $(r_0, \_) \notin \boldsymbol{cxS}$

$\mathrm{stR} \leftarrow \mathsf{U\text{-}ANON}_b.\mathrm{Expose}_R$
$\mathrm{xR} \leftarrow \mathtt{tru}$
Return stR

Oracle $\mathrm{ChallExpose}_S$

▷ If $\mathrm{xS} = \mathtt{tru} \vee \mathrm{cxS} = \mathtt{tru}$:
▷ Require $(s_0, \_) \notin \boldsymbol{cxS} \wedge (s_0, \_) \notin \boldsymbol{xS}$
◇ If $\mathrm{imp}_0 = \mathrm{imp}_1 = \mathtt{fal}$:
◇ Require $\mathrm{xR} = \mathrm{cxR} = \mathtt{fal}$

$\mathrm{stS}_b \leftarrow \mathsf{U\text{-}ANON}_b.\mathrm{ChallExpose}_S$
i If $b = 0$: $\boldsymbol{cstR}.\mathrm{append}(\boldsymbol{stR}[s_0])$
i If $b = 1$: $\boldsymbol{cstR}.\mathrm{append}(\mathsf{U\text{-}ANON}_1.\mathrm{ceStR})$
$\boldsymbol{cxS}.\mathrm{append}((s_0, s_1))$
$\mathrm{cxS} \leftarrow \mathtt{tru}$
Return stS_b

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

⊕ If $\mathrm{imp}_0 = \mathrm{imp}_1 = \mathtt{fal}$: Require $\mathrm{cxR} = \mathtt{fal}$
$(c, k) \xleftarrow{\$} \mathsf{U\text{-}ANON}_b.\mathrm{Snd}(\mathrm{ad})$
$\boldsymbol{cad}.\mathrm{append}(c, \mathrm{ad})$
$\boldsymbol{c} \xleftarrow{\cup} \{s_0\}$
$s_0 \xleftarrow{+} 1,\ s_1 \xleftarrow{+} 1$
i $(\boldsymbol{stR}[s_b], \_) \leftarrow \mathrm{RKE.rcv}(\boldsymbol{stR}[s_b - 1], c, \mathrm{ad})$
Return (c, k)

Oracle $\mathrm{ChallSnd}(\mathrm{ad})$

⊕ If $\mathrm{imp}_0 = \mathtt{fal}$: Require $\mathrm{xR} = \mathrm{cxR} = \mathtt{fal}$
$(c_b, k_b) \xleftarrow{\$} \mathsf{U\text{-}ANON}_b.\mathrm{ChallSnd}(\mathrm{ad})$
$\boldsymbol{cad}.\mathrm{append}(c_b, \mathrm{ad})$
$\boldsymbol{cc} \xleftarrow{\cup} \{s_0\}$
$s_0 \xleftarrow{+} 1$
i If $b = 0$: $(\boldsymbol{stR}[s_0], \_) \leftarrow \mathrm{RKE.rcv}(\boldsymbol{stR}[s_0 - 1], c_0, \mathrm{ad})$
Return (c_b, k_b)

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

$\mathrm{succ}_b \leftarrow \mathsf{U\text{-}ANON}_b.\mathrm{Rcv}(c, \mathrm{ad})$
If $\exists \hat{r} \geq \min(r_0, r_1)$ s.t. $(c, \mathrm{ad}) = \boldsymbol{cad}[\hat{r}]$
  If $b = 0$:
    $r'_1 \leftarrow \min(\boldsymbol{c} \setminus \boldsymbol{rcvd})$
    $\mathrm{succ}_1 \leftarrow \neg\mathrm{imp}_1 \wedge [\![r'_1 = \hat{r}]\!]$
    If succ_1: $\boldsymbol{rcvd} \xleftarrow{\cup} \{\hat{r}\}$
  If $b = 1$: $\mathrm{succ}_0 \leftarrow \neg\mathrm{imp}_0 \wedge [\![r_0 = \hat{r}]\!]$
  If succ_{1-b}: $r_{1-b} \xleftarrow{+} 1$
i Else: //check for impersonations
i Let $\mathcal{S} \subseteq \boldsymbol{xS}$ s.t. $(\_, r_1) \in \boldsymbol{xS}$
i If $|\mathcal{S}| > 1 \wedge (r_0, \_) \in \mathcal{S}$: unique $\leftarrow \mathtt{fal}$
i For $(\hat{r}_0, \hat{r}_1) \in \mathcal{S}$
i If $\mathrm{RKE.rcv}(\boldsymbol{stR}[\hat{r}_b], c, \mathrm{ad}) \neq (\_, \bot)$:
i $\mathrm{imp}_0 \leftarrow \mathrm{imp}_0 \vee [\![r_0 = \hat{r}_0]\!]$
i $\mathrm{imp}_1 \leftarrow \mathtt{tru}$
i If imp_{1-b}: $r_{1-b} \xleftarrow{+} 1$
i $\mathcal{I} \leftarrow \{i \mid \boldsymbol{cxS}[i] = (\hat{r}_0, \hat{r}_1)$ s.t. $\hat{r}_b = r_b\}$
i For $i \in \mathcal{I}$:
i If $\mathrm{RKE.rcv}(\boldsymbol{cstR}[i], c, \mathrm{ad}) \neq (\_, \bot)$:
i $\mathrm{imp}_0 \leftarrow \mathrm{imp}_0 \vee [\![r_0 = \hat{r}_0]\!]$, where $\hat{r}_0 = \boldsymbol{cxS}[i][0]$
i If imp_{1-b}: $r_{1-b} \xleftarrow{+} 1$
If succ_b: $r_b \xleftarrow{+} 1$
Return

Oracle $\mathrm{ChallExpose}_R$

◁ Require $\mathrm{xR} = \mathrm{cxR} = \mathtt{fal}$
◇ Require $(r_0, \_) \notin \boldsymbol{xS} \wedge (r_0, \_) \notin \boldsymbol{cxS}$
◇ Require $\mathrm{imp}_0 = \mathtt{fal}$
⊕ If $\mathrm{imp}_1 = \mathtt{fal}$: Require $s_0 = r_0$

$\mathrm{stR}_b \leftarrow \mathsf{U\text{-}ANON}_b.\mathrm{ChallExpose}_R$
$\mathrm{cxR} \leftarrow \mathtt{tru}$
Return stR_b

Setting: Game-Based Security Definitions

$\mathrm{SE.enc}(k, m) \to c$
$\mathrm{SE.dec}(k, c) \to m$

$\Pr[\mathrm{SE.dec}(k, \mathrm{SE.enc}(k, m)) = m] = 1$

$\underline{\mathrm{OW\text{-}CCA}_{\mathrm{SE}}(\mathcal{A}):}$
$M[\cdot] \leftarrow \bot$; $C \leftarrow \emptyset$
$k \leftarrow_{\$} \mathcal{K}$
$(c, m) \leftarrow_{\$} \mathcal{A}$
Stop with $M[c] = m$

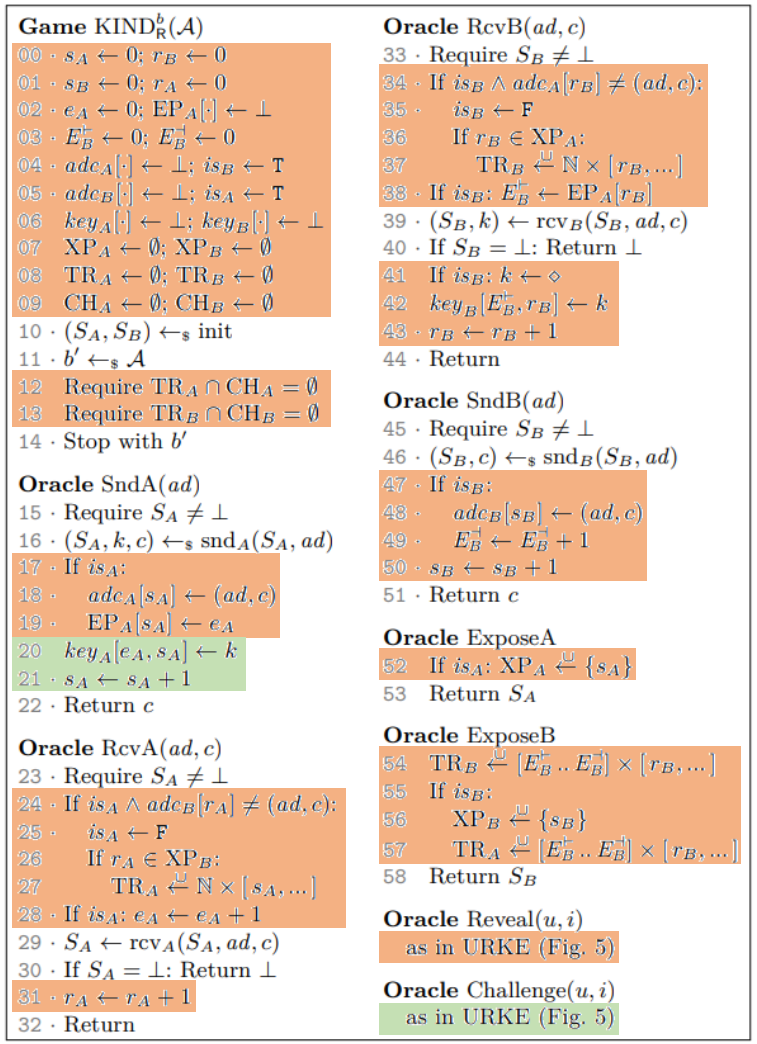
$\underline{\mathrm{Chall}():}$
$m \leftarrow_{\$} \mathcal{M}$
$c \leftarrow \mathrm{SE.enc}(k, m)$
$M[c] \leftarrow m$
$C \leftarrow C \cup \{c\}$
Return c

$\underline{\mathrm{Enc}(m):}$
Return $\mathrm{SE.enc}(k, m)$

$\underline{\mathrm{Dec}(c):}$
If $c \in C$ Return $\bot$
Return $\mathrm{SE.dec}(k, c)$

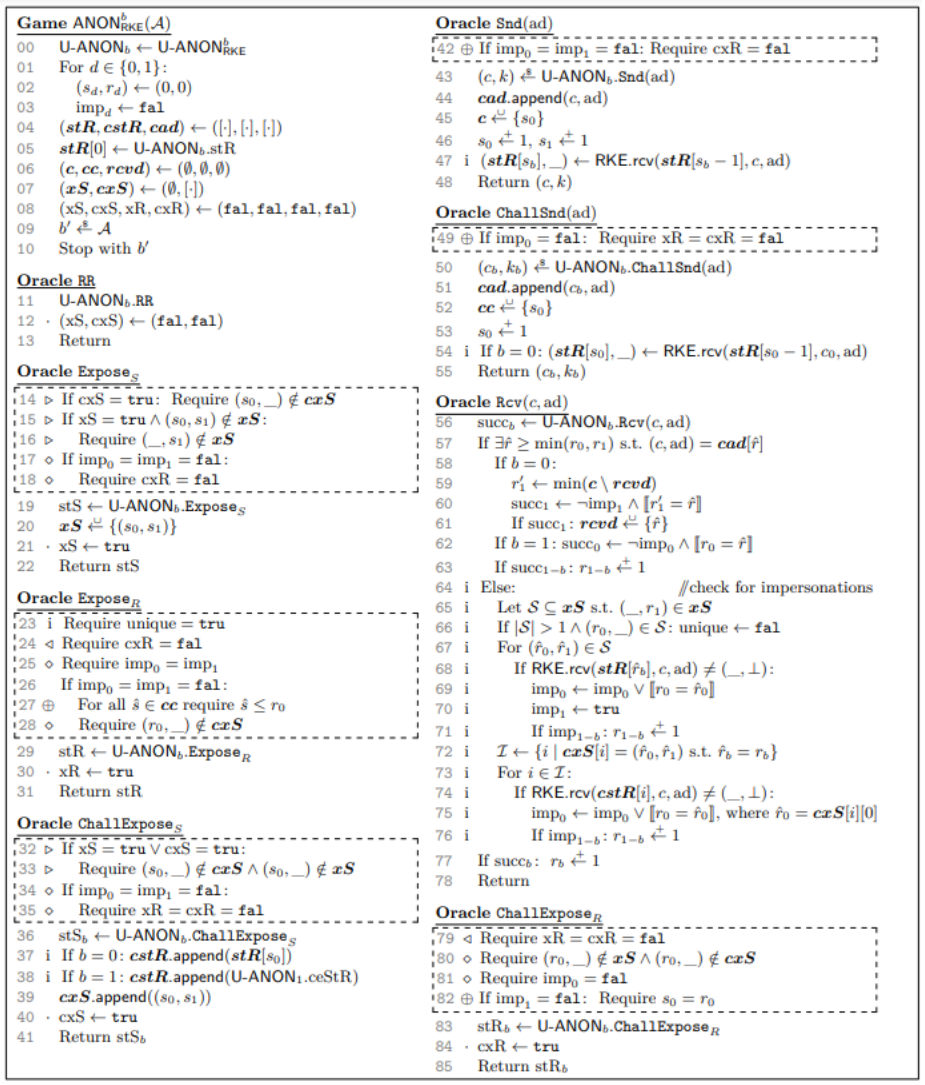
$\Pr[\mathrm{SE.dec}(k, \mathrm{SE.enc}(k, m)) = m] = 1$

Setting: Game-Based Security Definitions



- List of wrong definitions
  - Missed trivial attacks
- Validation via proof?!
  - JJ: Proof mistakes
- Worth of proofs?!
- Not sure we caught all…

→ Use tools!



Finding Trivial Attacks

- Syntax
- Correctness
- Adversarial goal
- Adversarial capabilities

} ⇒* Trivial attacks

Attacks against every construction

→No* algebraic structure

→Attacks against *ideal* construction

Simplifying Game-Based Definitions

Indistinguishability up to Correctness and Its Application to Stateful AE

Phillip Rogaway and Yusi Zhang

Computer Science Department
University of California Davis, One Shields Avenue, USA

Simplifying Game-Based Cryptographic Definitions

By

Yusi Zhang

DISSERTATION

Symbolic Perspective on OW-CCA

$$\mathbf{M} \mapsto \mathbf{K} \mid \mathrm{SE.enc}(\mathbf{K}, \mathbf{M})$$

$$m \in M \implies M \vdash m$$
$$M \vdash k, m \implies M \vdash \mathrm{SE.enc}(k, m)$$
$$M \vdash \mathrm{SE.enc}(k, m), k \implies M \vdash m$$

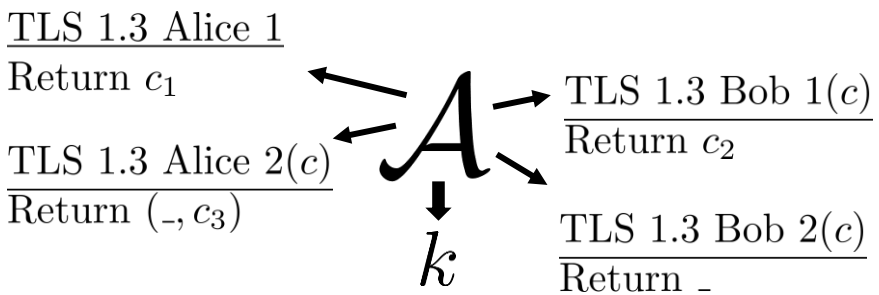
$\underline{\mathrm{OW\text{-}CCA}_{\mathrm{SE}}(\mathcal{A}):}$
$M[\cdot] \leftarrow \bot$
$k \leftarrow_{\$} \mathcal{K}$
$(\langle k, m \rangle, m) \leftarrow_{\$} \mathcal{A}$
Stop with $M[\langle k, m \rangle] = m$

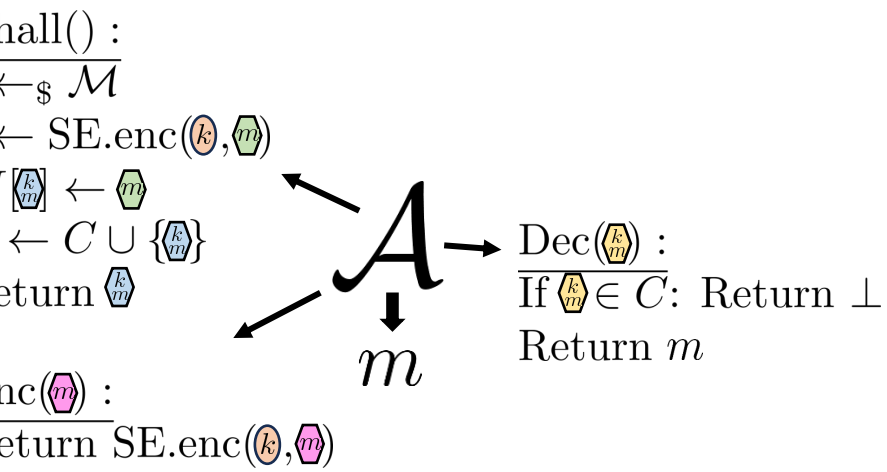
$\underline{\mathrm{Chall}():}$
$m \leftarrow_{\$} \mathcal{M}$
$\langle k, m \rangle \leftarrow \mathrm{SE.enc}(k, m)$
$M[\langle k, m \rangle] \leftarrow m$
$C \leftarrow C \cup \{\langle k, m \rangle\}$
Return $\langle k, m \rangle$

$\underline{\mathrm{Enc}(m):}$
Return $\mathrm{SE.enc}(k, m)$

$\underline{\mathrm{Dec}(\langle k, m \rangle):}$
If $\langle k, m \rangle \in C$: Return $\bot$
Return m

Symbolic Verification Tools





SoK: Computer-Aided Cryptography

Manuel Barbosa*, Gilles Barthe†‡, Karthik Bhargavan§, Bruno Blanchet§, Cas Cremers¶, Kevin Liao†‖, Bryan Parno**
*University of Porto (FCUP) and INESC TEC, †Max Planck Institute for Security & Privacy, ‡IMDEA Software Institute,
§INRIA Paris, ¶CISPA Helmholtz Center for Information Security, ‖MIT, **Carnegie Mellon University


```
theory symmEncOracles

begin
/* senc and sdec take 2 inputs, resp. */
functions: senc/2, sdec/2
/* decryption of encryption of m is m */
equations: sdec(k,senc(k,m))=m
/* ORACLES */
/* Samples a fresh challenge key and only permits a single challenge afterwards */
rule OracleInit:
  [ Fr(~challKey) ]
  --[ OnlyOnce() ]->
  [ !OracleInit(~challKey),
    PreChall() ]

/* Outputs the encryption of input message m under the challenge key */
rule OracleEnc:
  [ !OracleInit(~challKey),
    In(m) ]
  -->
  [ Out(senc(~challKey,m)) ]

/* Outputs decryption of input ciphertext under the challenge key as long as the challenge ciphertext has not been created yet */
/* For this, it requires that the SingleChallenge fact can be consumed and it produces this fact after it consumes it */
rule OracleDec1:
  [ !OracleInit(~challKey),
    PreChall(),
    In(senc(~challKey,m)) ]
  -->
  [ PreChall(),
    Out(m) ]

/* Ultimately consumes the SingleChallenge fact and outputs a challenge ciphertext */
rule OracleChall:
  [ !OracleInit(~challKey),
    PreChall(),
    Fr(~challMsg) ]
  -->
  [ !OracleChall(senc(~challKey,~challMsg)),
    Out(senc(~challKey,~challMsg)) ]

/* Outputs decryption of input ciphertext under the challenge key after the challenge ciphertext was created, unless the input is the challenge ciphertext */
/* For this, it requires that the input ciphertext differs from the challenge ciphertext */
rule OracleDec2:
  [ !OracleChall(senc(~challKey,~challMsg)),
    In(senc(~challKey,m)) ]
  --[ Neq(senc(~challKey,~challMsg),senc(~challKey,m)) ]->
  /*-->*/ /*For sanity check: removing limitation of challenge decryption */
  [ Out(m) ]

/* Specifies the final query of the adversary to which it provides the challenge message; if the input message equals the challenge message, the adversary wins */
rule OracleFin:
  [ !OracleChall(senc(~challKey,~challMsg)),
    In(m) ]
  --[ Eq(m,~challMsg),
      !AdvWins() ]->
  [ ]

restriction Equality:
  "All x y #i. Eq(x,y) @#i ==> x = y"
restriction Inequality:
  "All x #i. Neq(x,x) @ #i ==> F"
restriction OnlyOnce:
  "All #i #j. OnlyOnce()@#i & OnlyOnce()@#j ==> #i = #j"

/* Lemma that is impossible to obtain ~challMsg" */
lemma oneWaySecurity:
  " not( Ex #i. !AdvWins() @i ) "

end
```

Tool		Unbound	Trace	Equiv	Eq-thy	State	Link
CPSA$^{\triangleright}$	[16]	●	●	○	○	●	○
F7$^{\diamond}$	[17]	●	●	○	◐	●	●
↳F5$^{\diamond}$	[18]	●	●	○	◐	●	●
Maude-NPA$^{\triangleright}$	[19]	●	●	●^{d}	●	○	○
ProVerif*†	[20]	●	●	●^{d}	◐	○	○
↳fs2pv$^{\diamond\dagger}$	[21]	●	●	○	◐	○	●
↳GSVerif*†	[22]	●	●	○	◐	●	○
↳ProVerif-ATP*†	[23]	●	●	○	●	○	○
↳StatVerif*†	[24]	●	●	●^{d}	◐	●	○
Scyther$^{\triangleright}$	[25]	●	●	○	○	○	○
scyther-proof$^{\triangleright\ddagger\S}$	[26]	●	●	○	○	○	○
Tamarin*‡	[27]	●	●	●^{d}	●	●	○
↳SAPIC*	[28]	●	●	○	●	●	○
CI-AtSe$^{\triangleright}$	[29]	○	●	○	●	●	○
OFMC$^{\triangleright\dagger}$	[30]	○	●	○	◐	●	○
SATMC$^{\triangleright}$	[31]	○	●	○	○	●	○
AKISS*	[32]	○	○	●^{t}	●	●	○
APTE*	[33]	○	○	●^{t}	○	●	○
DEEPSEC*	[34]	○	○	●^{t}	◐	●	○
SAT-Equiv*	[35]	○	○	●^{t}	○	○	○
SPEC*,§	[36]	○	○	●^{o}	○	○	○

Specification language	Miscellaneous symbols
▷ – security protocol notation	↳ – previous tool extension
⋆ – process calculus	† – abstractions
∗ – multiset rewriting	‡ – interactive mode
◇ – general programming language	§ – independent verifiability

Equational theories (Eq-thy)	Equivalence properties (Equiv)
● – with AC axioms	t – trace equivalence
◐ – without AC axioms	o – open bisimilarity
○ – fixed	d – diff equivalence

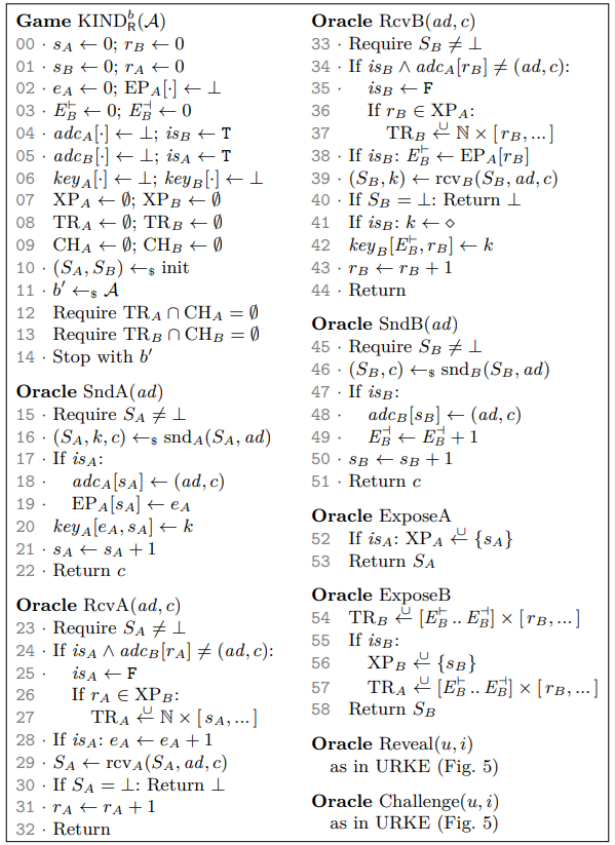
TABLE I
OVERVIEW OF TOOLS FOR SYMBOLIC SECURITY ANALYSIS. SEE SECTION II-B FOR MORE DETAILS ON COMPARISON CRITERIA.

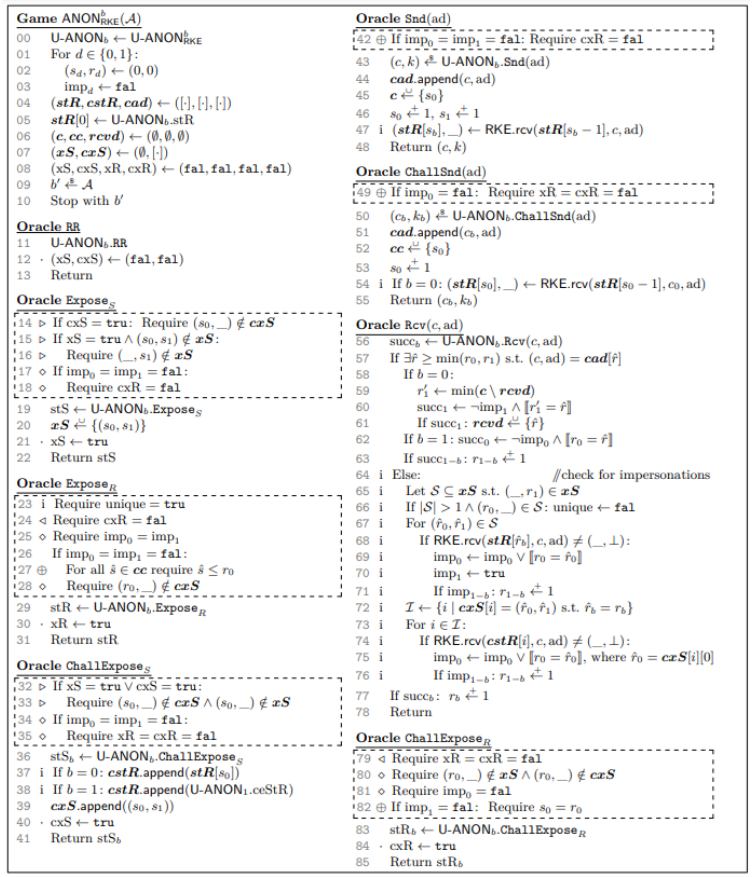
Automated Attack Search

- OW-CCA: SKE, PKE, KEM
- SUF-CMA: MAC, DS

- Correctness:
  - Statefulness
  - Loops
  - …

- Games:
  - …





Actual Idea Direction

- Implement definitions
- Verify definitions / Reproduce attacks
- Prove: Symbolic attacks ⇒ Trivial attacks
- Prove: No symbolic attacks ⇒ No trivial attacks

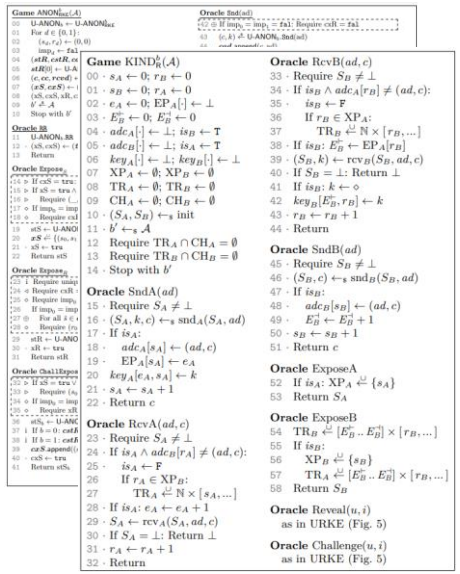
$\underline{\text{OW-CCA}_{\text{SE}}(\mathcal{A}):}$
$M[\cdot] \leftarrow \bot$
$k \leftarrow_{\$} \mathcal{K}$
$(k\|m, m) \leftarrow_{\$} \mathcal{A}$
Stop with $M[k\|m] = m$

$\underline{\text{Chall}():}$
$m \leftarrow_{\$} \mathcal{M}$
$k\|m \leftarrow \text{SE.enc}(k, m)$
$M[k\|m] \leftarrow m$
$C \leftarrow C \cup \{k\|m\}$
Return $k\|m$

$\underline{\text{Enc}(m):}$
Return $\text{SE.enc}(k, m)$

$\underline{\text{Dec}(k\|m):}$
If $k\|m \in C$: Return $\bot$
Return m

- WIP:
  - Feedback
  - Discussion
  - Happy to collaborate
- Q:
  - Which tools?
  - Discussing proofs
  - Simple fail examples



Tool		Unbound	Trace	Equiv	Eq-thy	State	Link
CPSA▷	[16]	●	●	○	○	●	○
F7◇	[17]	●	●	○	◐	●	●
↳F5◇	[18]	●	●	○	◐	●	●
Maude-NPA▷	[19]	●	●	●[d]	●	○	○
ProVerif⋆†	[20]	●	●	●[d]	◐	○	○
↳fs2pv◇†	[21]	●	●	○	◐	○	●
↳GSVerif⋆†	[22]	●	●	○	◐	●	○
↳ProVerif-ATP⋆†	[23]	●	●	○	●	○	○
↳StatVerif⋆†	[24]	●	●	●[d]	◐	●	○
Scyther▷	[25]	●	●	○	○	○	○
scyther-proof▷‡§	[26]	●	●	○	○	○	○
Tamarin⋆‡	[27]	●	●	●[d]	●	●	○
↳SAPIC⋆	[28]	●	●	○	●	●	○
CI-AtSe▷	[29]	○	●	○	●	●	○
OFMC▷†	[30]	○	●	○	◐	●	○
SATMC▷	[31]	○	●	○	○	●	○
AKISS⋆	[32]	○	○	●	●	●	○
APTE⋆	[33]	○	○	●[d]	◐	●	○
DEEPSEC⋆	[34]	○	○	●[d]	◐	●	○
SAT-Equiv⋆	[35]	○	○	●[d]	○	○	○
SPEC⋆·§	[36]	○	○	●[o]	○	○	○

Specification language
▷ – security protocol notation
⋆ – process calculus
∗ – multiset rewriting
◇ – general programming language

Miscellaneous symbols
↳ – previous tool extension
† – abstractions
‡ – interactive mode
§ – independent verifiability

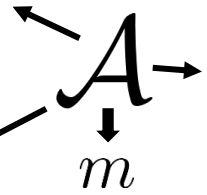
Equational theories (Eq-thy)
● – with AC axioms
◐ – without AC axioms
○ – fixed

Equivalence properties (Equiv)
t – trace equivalence
o – open bisimilarity
d – diff equivalence

TABLE I
OVERVIEW OF TOOLS FOR SYMBOLIC SECURITY ANALYSIS. SEE SECTION II-B FOR MORE DETAILS ON COMPARISON CRITERIA.

$\underline{\text{Chall}():}$
$m \leftarrow_{\$} \mathcal{M}$
$k\|m \leftarrow \text{SE.enc}(k, m)$
$M[k\|m] \leftarrow m$
$C \leftarrow C \cup \{k\|m\}$
Return $k\|m$

$\underline{\text{Enc}(m):}$
Return $\text{SE.enc}(k, m)$



$\underline{\text{Dec}(k\|m):}$
If $k\|m \in C$: Return $\bot$
Return m