Composable security

Supervisor: Markulf Kohlweiss
Ruxandra Icleanu

University of Edinburgh
LFCS Summer 2023

Overview

- Desiderata
- Approaches
- Ex 1: Secure channel & authenticated channel
- Ex 2: Commit reveal
- Other frameworks
- UC?

Desiderata

We want a framework for analysing the security of cryptographic protocols that would ideally satisfy two properties:

(1) Composability
If protocols P_1 and P_2 are secure, we want to be able to directly conclude that $P_1 \circ P_2 = P_3$ is also secure.
Note:
- There are many frameworks for which this does not hold.
- We need to define the composition operation.

(2) Modularity
It should be enough to prove the security of some 'building blocks'.

Universally composable security

But don't we already have such a framework?
UC security (Canetti, 2000)

Universally composable security

But don't we already have such a framework?
UC security (Canetti, 2000)

(1) $\checkmark$

(2) $\sim$

Problem: Proofs are done using interactive Turing machines (ITMs). In practice, it is difficult to use this computation model, and proofs end up being very informal.

Approaches

What can we do?

Approaches

What can we do?

- Formalise proofs for a few important protocols

Approaches

What can we do?

- Formalise proofs for a few important protocols
- Find a model equivalent to ITMs

Approaches

What can we do?

- Formalise proofs for a few important protocols
- Find a model equivalent to ITMs
- Use a new framework

Approach 1

[Summer 2023] We wrote pseudocode for UC ideal functionalities in a fashion that is more aligned with game-based techniques.

Ex 1: AC + SK $\equiv$ SC

Recall:

- secure channel: the adversary can eavesdrop to the ciphertext, but cannot tamper with it
- authenticated channel: the adversary is able to both eavesdrop and to tamper with it

Ex 1: AC + SK $\equiv$ SC

Recall:

- secure channel: the adversary can eavesdrop to the ciphertext, but cannot tamper with it
- authenticated channel: the adversary is able to both eavesdrop and to tamper with it

Assumptions:

(i) no distinction between environment and adversary (equivalently, the adversary has full capabilities)

(ii) the adversary can influence if the last message sent will be delivered

(iii) the length of the message is leaked

Ex 1: AC + SK $\equiv$ SC

Recall:

- secure channel: the adversary can eavesdrop to the ciphertext, but cannot tamper with it
- authenticated channel: the adversary is able to both eavesdrop and to tamper with it

Assumptions:

(i) no distinction between environment and adversary (equivalently, the adversary has full capabilities)

(ii) the adversary can influence if the last message sent will be delivered

(iii) the length of the message is leaked

Goals:

- write formal descriptions of the authenticated channel and secure channel models

Ex 1: AC + SK $\equiv$ SC

Recall:

- secure channel: the adversary can eavesdrop to the ciphertext, but cannot tamper with it
- authenticated channel: the adversary is able to both eavesdrop and to tamper with it

Assumptions:

(i) no distinction between environment and adversary (equivalently, the adversary has full capabilities)

(ii) the adversary can influence if the last message sent will be delivered

(iii) the length of the message is leaked

Goals:

- write formal descriptions of the authenticated channel and secure channel models
- show that the authenticated channel together with a secret key can be transformed into a secure channel via one-time pad encryption

Ex 1: AC + SK $\equiv$ SC

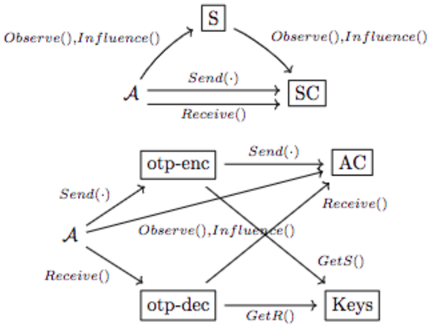


Figure 1: SC and AC models

Ex 1: AC + SK $\equiv$ SC

SC

```
msg[] ← ⊥
deliver[] ← ⊤
ctrS_SC, ctrR_SC ← 0
Send(m)
    msg[ctrS_SC] ← m
    ctrS_SC ← ctrS_SC + 1
    return ⊥
Receive()
    ctrR_SC ← ctrR_SC + 1
    if deliver[ctrR_SC - 1] = ⊥
        return ⊥
    return msg[ctrR_SC - 1]
Observe()
    return ctrS_SC
Influence()
    deliver[ctrS_SC - 1] ← ⊥
    return ⊥
```

Ex 1: AC + SK $\equiv$ SC

S

```
fakeMsg[] ← ⊥
Observe()
    lengthSend ← SC.Observe()
    lengthFaked = fakeMsg.length
    // length is the largest index with a non ⊥ value +1
    for i in lengthFaked .. lengthSend-1
        fakeMsg[i] ← {0, 1}^n
    return fakeMsg[] // returns all messages faked so far
Influence()
    return SC.Influence()
```

Ex 1: AC + SK ≡ SC

Keys

```
keys[] ← ⊥
ctrS ← 0
ctrR ← 0
keys[] ← ⊥
GetS()
    keys[ctrS] ← {0, 1}^n
    ctrS ← ctrS + 1
    return keys[ctrS - 1]
GetR()
    ctrR ← ctrR + 1
    return keys[ctrR - 1]
```

Ex 1: $AC + SK \equiv SC$

otp-enc

```
Send(m)
    k ← GetS()
    AC.Send(m ⊕ k)
    return ⊥
```

otp-dec

```
Receive()
    k ← GetR()
    c ← AC.Receive()
    return c ⊕ k
```

Ex 1: AC + SK $\equiv$ SC

$\mathtt{AC}_{sid=(i,j,ctr)}$

```
msg[] ← ⊥
deliver[] ← ⊤
ctrS_AC, ctrR_AC ← 0
i.Send_sid(m)
    msg[ctrS_AC] ← m
    ctrS_AC ← ctrS_AC + 1
    return ⊥
j.Receive_sid()
    ctrR_AC ← ctrR_AC + 1
    if deliver[ctrR_AC - 1] = ⊥
        return ⊥
    return msg[ctrR_AC - 1]
Observe_sid()
    return msg[] // returns all messages sent so far
Influence_sid()
    deliver[ctrS_AC - 1] ← ⊥
    return ⊥
```

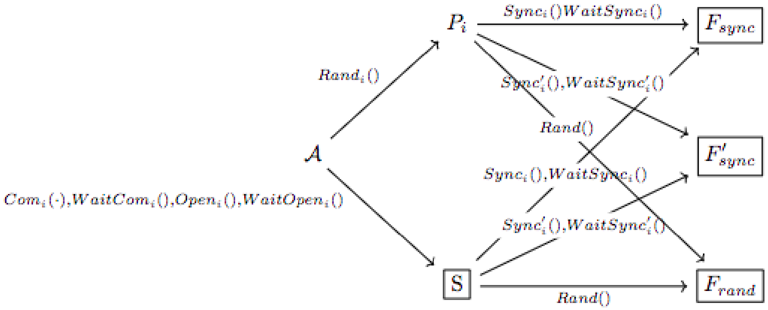
Ex 2: Commit reveal

- n-party random protocol
- idea: each of the n parties draws a value uniformly at random, commits to the value and waits for everyone to do so, then opens the value and waits for everyone to do so; the values are then summed up
- the final value is random and unbiased

Goals:

- describe the real protocol and an ideal protocol + simulator
- understand why the two are equivalent
- motivate our choice of simulator

Ex 2: Commit reveal



Ex 2: Commit reveal

We use Lucas Meier's "Towards Modular Protocol Security (and beyond?)" COSIC Seminar talk as a starting point.

We define the real protocol as

$$\mathcal{P}_{rand} = \bigotimes_{i=1}^{n} P_i \diamond F_{commit}$$

where:

P_i

$\underline{\texttt{Rand}_i()}$

$\quad \texttt{x} \xleftarrow{\$} \mathbb{Z}/m\mathbb{Z}$
$\quad \texttt{Com}_i\texttt{(x)}$
$\quad \texttt{WaitCom}_i\texttt{()}$
$\quad \texttt{Open}_i\texttt{()}$
$\quad \vec{x} \leftarrow \texttt{WaitOpen}_i\texttt{()}$
$\quad \texttt{return } \sum_{j=1}^{n} x_j$

Ex 2: Commit reveal

F_{commit}

```
Com_i(x)
    if x_i ≠ ⊥
        return ⊥
    yield(⟨ Com_i() ⟩)
    x_i ← x
    return ()
WaitCom_i()
    wait ∀ j . x_j ≠ ⊥
    return ()
Open_i()
    if x_i = ⊥
        return ⊥
    yield(⟨ Open_i() ⟩)
    open_i ← ⊤
    return ()
WaitOpen_i()
    wait ∀ j . open_j
    return x⃗
```

Ex 2: Commit reveal

We define the ideal protocol as:

$$\mathcal{P}_{ideal-rand} = P_i \diamond (F_{sync} \otimes F'_{sync} \otimes F_{rand})$$

where:

P_i

```
Rand_i()
    Sync_i()
    WaitSync_i()
    Sync'_i()
    WaitSync'_i()
    return Rand()
```

Ex 2: Commit reveal

F_{sync}

```
sync_i ← ⊥
Sync_i()
    yield(⟨ Sync_i() ⟩)
    sync_i ← ⊤
    return ()
WaitSync_i()
    wait ∀ j . sync_j
    return ()
```

F'_{sync}

```
sync'_i ← ⊥
Sync'_i()
    yield(⟨ Sync'_i() ⟩)
    sync'_i ← ⊤
    return ()
WaitSync'_i()
    wait ∀ j . sync'_j
    return ()
```

Ex 2: Commit reveal

Notation: $\mathcal{M}$ = the set of malicious parties
$\mathcal{H} = \{P_1, .., P_n\} \setminus \mathcal{M}$ the set of honest parties
We use index i for an honest party,
and index k for a malicious one

S

$\vec{x} \leftarrow \bot$

<u>$\mathtt{Com}_k(\mathtt{x})$</u>
- if $x_k \neq \bot$
  - return $\bot$
- $x_k \leftarrow x$
- $\mathtt{Sync}_k()$
- return ()

<u>$\mathtt{WaitCom}_k()$</u>
- return $\mathtt{WaitSync}_k()$

<u>$\mathtt{Open}_k()$</u>
- if $\mathtt{x}_k = \bot$
  - return $\bot$
- return $\mathtt{Sync}'_k()$

...

Ex 2: Commit reveal

S

```
...
WaitOpen_k()
    WaitSync'_k()
    j <-$- H // fix some honest party
    for i ∈ H \ {k}
        // need to ensure all parties will have the same vector x⃗
        if x_i ≠ ⊥
            x_i <-$- Z/mZ
    x_j <- Rand() - Σ_{i ≠ k} x_i
    return x⃗
handle yield(⟨ Sync_i() ⟩; c)
    yield(⟨ Com_i() ⟩)
    c()
handle yield(⟨ Sync'_i() ⟩; c)
    yield(⟨ Open_i() ⟩)
    c()
```

Ex 2: Commit reveal

Alternative simulators?

- S' in which all honest parties are sampled uniformly at random
  Problem: the values in $\overrightarrow{x}$ will not necessarily sum up to the value given by `Rand()`
  Attack: on `WaitOpen()`

Ex 2: Commit reveal

Alternative simulators?

- S' in which all honest parties are sampled uniformly at random
  Problem: the values in $\overrightarrow{x}$ will not necessarily sum up to the value given by `Rand()`
  Attack: on `WaitOpen()`
- S'' which doesn't check that a party has committed to a value before opening the value

Ex 2: Commit reveal

Alternative simulators?

- S' in which all honest parties are sampled uniformly at random
  Problem: the values in $\overrightarrow{x}$ will not necessarily sum up to the value given by `Rand()`
  Attack: on `WaitOpen()`
- S'' which doesn't check that a party has committed to a value before opening the value
- S''' which doesn't use F'_{sync}

Other examples

- presence check protocol
- NIZK

Approach 2: Other frameworks

- Modular Protocol Security (MPS)
  - builds on state-separable proofs
  - two ways of 'combining' protocols: tensoring $\otimes$ and composition $\triangleleft$
  - simulation between protocols (instead of simulation between protocol and an ideal functionality)
- Constructive cryptography
- Synthetic cryptography

Approach 3: UC?

What if we just stay in UC, but find a workaround?
Would want a computational model that is equivalent to ITMs, but easier to use.

- Yanofsky (2022): category of TMs can be shown to be equivalent to the category of computable functions
- Can we extend this idea for ITMs?

Approach 3: UC?

What if we just stay in UC, but find a workaround?
Would want a computational model that is equivalent to ITMs, but easier to use.

- Yanofsky (2022): category of TMs can be shown to be equivalent to the category of computable functions
- Can we extend this idea for ITMs? Possibly ..

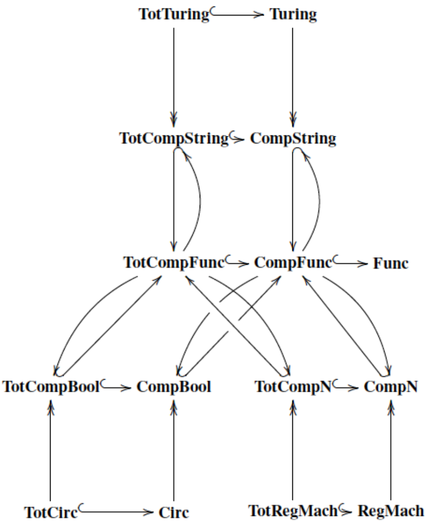


Figure: Big picture of models

(from Yanofsky's "Theoretical Computer Science for the Working Category Theorist)

References

[1] Mike Rosulek - *The Joy of Cryptography*

[2] Lúcás Meier - *Towards Modular Foundations for Protocol Security*

[3] Noson S. Yanofsky (2022) - *"Theoretical Computer Science for the Working Category Theorist"*

[4] Chris Brzuska, Antoine Delignat-Lavaud, Cédric Fournet, Konrad Kohbrok, Markulf Kohlweiss - *State separation for code-based game-playing proofs*

[5] Ran Canetti - Universally Composable Security: A New Paradigm for Cryptographic Protocols

[6] Ueli Maurer - *Constructive Cryptography – A New Paradigm for Security Definitions and Proofs*