On the composability, modularity, and security of cryptographic protocols

Ran Canetti
IBM Research
Cryptographic protocols are basic building-blocks in fault-tolerant systems.

Examples:

• Secure communication:
  – Key exchange
  – Encryption (symmetric, asymmetric)
  – Digital signatures, authentication codes

• Agreement and broadcast:
  – Joint coin tossing
  – Secret sharing
  – Signatures
Cryptographic protocols are basic building-blocks in fault-tolerant systems.

More examples:

• Secure distributed depositories and services:
  – Threshold signatures, encryption
  – Secret sharing

• Secure and private information retrieval

• …
Analysis of cryptographic protocols is challenging:

• Security properties are not absolute:
  – Based on computational hardness assumptions (rather than being unconditional)
  – Involve probabilities of error

• Properties are complex to state, prove, and interpret.

• Properties are context-dependent (protocols may interact badly with each other)
Consequently:

Systems that involve cryptographic protocols are hard to analyze. This holds even if the crypto is a “small part” of the overall system.

How to model and analyze such systems?
The formal methods approach: “abstract out” the crypto

- Analyze protocols in an abstract model where the crypto primitives are “ideal boxes”. E.g.:
  - Encryption provides absolute, tamper-proof secrecy and integrity [Dolev-Yao83]
  - Communication over a secure channel is completely unseen/untamperable by third parties [Abadi-Gordon93]
- “Hope” that the abstract protocols remain secure when realized using “real crypto”.
Advantages of the “formal methods approach”:
  – Simplifies the analysis (no computational issues)
  – Separates the “crypto part” from the “non-crypto” part

Disadvantages:
  – Does not address potential flaws in, and bad interactions with, the “crypto part”
  – Does not guarantee security of the “real protocol”
The traditional cryptographic approach:

- Adversary is a computational entity (resource-bounded Turing machine).
- Has access to the “real” bit-strings of communication.
- Security properties are formulated accordingly.
Advantage of the cryptographic approach:

– Guarantees security of “real protocols”

Disadvantages:

– Complex to state, prove, interpret…
– Cryptographic protocols do not “compose”
– Requires cryptographic modeling of the entire system, even if the crypto is only a small part.
General Goal

Would like to analyze fault-tolerant systems in a modular way:

• Represent the cryptographic parts as “ideal boxes”.

• Analyze the overall system assuming access to the “ideal crypto boxes”.

• Realize the “ideal crypto boxes” using real cryptographic protocols

• Deduce the security of the overall, “composed” system.
A framework for “universally composable security” \[C01\]

- Security of cryptographic protocols is defined via realizing an “idealized trusted service”
- A central property: protocols can be composed in a very general way while maintaining security.

Can be used to carry out the “modular analysis” approach.

Similar framework defined in [Pfitzmann-Waidner00,01]
Rest of the talk:

- Present the notion of security
- Present the composition theorem
- Discuss usage
The general definitional approach

[Goldreich-Micali-Wigderson87]

‘A protocol is secure for some task if it “emulates” an “ideal setting” where the parties hand their inputs to a “trusted party”, who locally computes the desired outputs and hands them back to the parties.’

Several formalizations of this fundamental approach exist (e.g. [Goldwasser-Levin90, Micali-Rogaway91, Beaver91, Canetti93, Pfitzmann-Waidner94, Canetti00, Dodis-Micali00, Pfitzmann-Schunter-Waidner00]), But:

– Only limited “secure composition” guarantees
– Typically restricted to “function evaluation”
The general definitional approach
[Goldreich-Micali-Wigderson87]

‘A protocol is secure for some task if it “emulates” an “ideal setting” where the parties hand their inputs to a “trusted party”, who locally computes the desired outputs and hands them back to the parties.’

Several formalizations of this fundamental approach exist (e.g. [Goldwasser-Levin90, Micali-Rogaway91, Beaver91, Canetti93, Pfitzmann-Waidner94, Canetti00, Dodis-Micali00, Pfitzmann-Schunter-Waidner00]), But:

– Only limited “secure composition” guarantees
– Typically restricted to “function evaluation”
How security is defined (I):

1. Write an “ideal functionality” $F$ that captures the requirements of the task at hand.

   $F$ is a “code for an ideal trusted service on the net”. ($F$ Captures both correctness and secrecy requirements.)
Example:
The authenticated message transmission functionality

- Receive (A,B,m) from A
- Send (A,B,m) to B and the adversary, and halt.
Example:
The secure message transmission functionality

- Receive \((A,B,m)\) from A
- Send \((A,B,m)\) to B, send \((A,B,|m|)\) to the adversary, and halt.
Example:
The key-exchange functionality $F_{KE}$

- Upon receiving ("exchange",A,B,sid) from parties A and B, do:
  - choose a random key $k$.
  - send $k$ to A and B.
  - send (A,B,sid) to the adversary.
  - Halt.

(If either party asks to set the key to some value then $F_{KE}$ agrees.)
Example:
The ZK functionality (for relation R)

- Receive (P,V,x,w) from P
- Receive (V,P,x) from V
- Send (R(x,w)) to V and halt.

Note:
- V is assured that it accepts only if R(x,w)=1 (soundness)
- P is assured that V learns nothing but R(x,w) (Zero-Knowledge)
How security is defined (II):

2. Say that a protocol $\pi$ *emulates* the ideal process for evaluating F if no “external environment” Z can tell between:

- A run of protocol $\pi$.
- An “ideal execution” where the parties interact with F.

(In this case, we say that $\pi$ *securely realizes* functionality F.)
A bit more precisely:

Ideal process:

Protocol execution:
A bit more precisely:

Ideal process:

Protocol execution:

Protocol $\pi$ securely realizes $F$ if:

For any adversary $A$
There exists an adversary $S$
Such that no environment $Z$ can tell whether it interacts with:
- A run of $\pi$ with $A$
- An ideal run with $F$ and $S$
Universal Composition:

1. Present the composition operation

2. State the composition theorem
The composition operation
(Originates with [Micali-Rogaway91])

Start with:
• Protocol $\rho^F$ that uses ideal calls to $F$
• Protocol $\pi$ that securely realizes $F$

Construct the composed protocol $\rho^\pi$:
• Each call to $F$ is replaced with an invocation of $\pi$.
• Each value returned from $\pi$ is treated as coming from $F$.

Note: In $\rho^F$ parties call many copies of $F$.
⇒ In $\rho^\pi$ many copies of $\pi$ run concurrently.
The composition operation 
(single call to F)
The composition operation
(single call to F)
The composition operation
(multiple calls to F)
The universal composition theorem: [C. 01]

Protocol $\rho^\pi$ “emulates” protocol $\rho^F$.

(That is, for any adversary A there exists an adversary A` such that no Z can tell whether it is interacting with $(\rho^\pi, A)$ or with $(\rho^F, A`)$.)

Corollary: If $\rho^F$ securely realizes functionality G then so does $\rho^\pi$

(Weaker composition theorems were proven in e.g. [Micali-Rogaway91, Canetti00, Dodis-Micali00, Pfitzmann-Schunter-Waidner00].)
Implications of the UC theorem

- Can design and analyze protocols in a modular way:
  - Partition a given task $T$ to simpler sub-tasks $T_1 \ldots T_k$
  - Construct protocols for realizing $T_1 \ldots T_k$.
  - Construct a protocol for $T$ assuming ideal access to $T_1 \ldots T_k$.
  - Use the composition theorem to obtain a protocol for $T$ from scratch.

  (Analogous to subroutine composition for correctness of programs, but with an added security guarantee.)
Implications of the UC theorem

• Assume protocol $\pi$ securely realizes ideal functionality $F$. Can deduce security of $\pi$ in any multi-execution environment:

As far as the environment is concerned, interacting with (multiple copies of) $\pi$ is equivalent to interacting with (multiple copies of) $F$. 
Formulating “ideal crypto boxes” within the UC framework

- Write ideal functionalities that capture security properties of known primitives
- Show that the functionalities can be realized via cryptographic protocols
- Can now analyze protocols assuming access to the ideal functionalities. 
  (this is often doable without getting into computational issues)
Was done for:

- Digital signatures
- Public-key encryption
- Key exchange
- Secure communication
- Two-party protocols (commitment, ZK, oblivious transfer, coin tossing,...)
- General multi-party functionalities

(Work done in C01, Pfitzmann-Waidner01, C-Fischlin01, C-Lindell-Ostrovsky-Sahai01, C-Krawczyk02, Backes-Pfitzmann-Waidner03,...)
Two approaches for writing functionalities

• Approach 1: Have multiple copies of simple ideal functionalities [C01].
  – Define separate functionalities for encryption, signature, authentication, key exchange, etc.
  – Each functionality for a single instance.
• Approach 2: Have a single “monolithic” ideal functionality that represents all the crypto [BPW03].
  – A single functionality captures all instances of all crypto primitives used in the system.
Future work

• Write ideal functionalities for representing more cryptographic primitives.
• Prove security of more protocols in the UC framework.
• Design formal tools for analyzing security of protocols within the UC framework, assuming ideal access to crypto primitives.

End goal: Automated, cryptographically sound analysis.
General goal:

Would like to combine the two analytical approaches, to get “the best of both worlds”.

(First attempts done by [Abadi-Rogaway01, ...])