How To Obtain and Assert Composable Security

Ran Canetti
IBM Research


Nice sun...


yeah...


You know, I lost more than you in the stock market.


No way. How much did you lose?

I won’t tell you…
How much did
you lose?


You tell first!


No,
you tell first!


No,
you tell first!

I lost
X$

I lost
Y$

is
X>Y?

The millionaires problem [Yao82]

I lost
X$

I lost
Y$

is
X>Y?

Cryptographic tasks

Two or more parties want to perform some joint computation, while guaranteeing "security" against "adversarial behavior".

Some Cryptographic applications

- Secure communication:
  - Secure Communication Sessions
  - Virtual Private Networks
  - Secure Email
- Secure storage:
  - Secure Remote Storage
  - Secure peer-to-peer systems
- "E-commerce":
  - Auctions, trading and financial markets,
  - Shopping
- Database Security:
  - Private information retrieval, Database pooling, Privacy
- Electronic voting
- On-line gambling …

Some basic cryptographic building blocks

- Key Exchange [Diffie-Hellman78]
- Contract Signing/Fair Exchange [Even-Goldreich-Lempel85]
- Coin-tossing [Blum82]
- Zero-Knowledge [Goldwasser-Micali-Rackoff88]
- Commitment [Blum88]
- Oblivious Tranfer [Rabin81]
- Secret Sharing [Shamir79]
- …

A plethora of cryptographic protocols

Many cryptographic protocols were developed over the years:

- Obtaining authenticated and secure communication [DH78,Needham-Schroeder78,Bird+91,Bellare-Rogaway93, Kerberos, PGP,SSL/TLS,IPSEC,…]
- General constructions: Can “securely carry out” any cryptographic task, given authenticated communication [Y86,GMW87,BGW88,RB89,…]
- More efficient constructions for specific problems

What does “security” mean?

Some concerns:

- Correctness of local outputs:
  - As a function of all inputs
  - Also distributional and unpredictability guarantees

  [e.g., entity/input authentication, tally correctness, input independence, unbiased randomness of ouput.]
- Secrecy of local data and inputs
- Privacy
- Fairness
- Accountability
- Availability

However, rigorously capturing the intuitive notion of security is a tricky business…

Main stumbling points:

- Security can often hold only against computationally bounded adversaries and only in a probabilistic sense
- Unexpected inter-dependencies between security requirements
- Unexpected “bad interference” between different protocol instances in a system

In the rest of this talk:

- Demonstrate the problem
- Describe a paradigm for formulating definitions of security, in a way that guarantees security in any execution environment
- Review some results within this paradigm

Insufficiency of stand-alone security

1st example: Sharing Keying Material

A simple insecure protocol combination

Let π be some “really secure” protocol where the parties use an n-bit secret key k. Define:

- Protocol π_1:
  - Parties use a 2n-bit key $k=k_1k_2$.
  - Publicize k_1; run π on k_2.
- Protocol π_2:
  - Parties use a 2n-bit key $k=k_1k_2$.
  - Publicize k_2; run π on k_1.

A simple insecure protocol combination

Observe:

- When run alone, both π_1 and π_2 are just as secure as π.
- As soon as π_1 and π_2 are run together, they both become completely insecure.

(Similar examples given in [Kelsey,Schneier,Wagner 97])

The problem: The two protocols use joint secret information in an “uncoordinated way”.

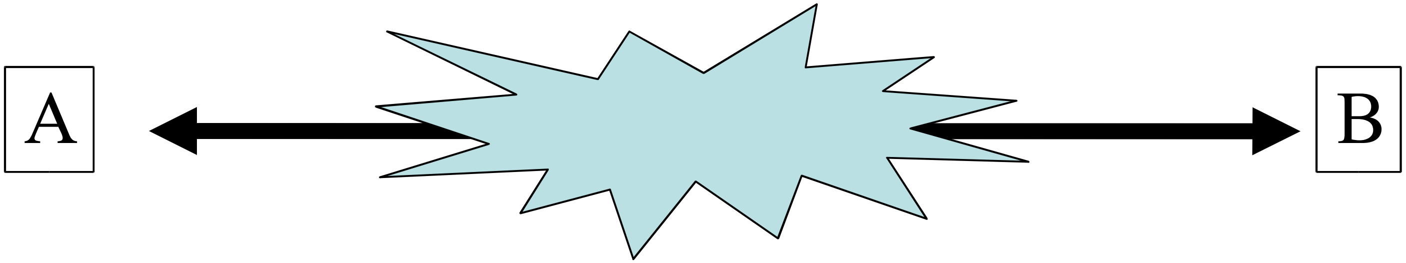
Perhaps if we rule out such cases we'll be ok?

Insufficiency of stand-alone security

2nd example: Key-Exchange and secure communication

Authenticated Key Exchange

The goal: Two parties want to generate a common, random and secret key over an untrusted network.



- The main use is to set up a secure communication session: Each message is encrypted and authenticated using the generated key.

The basic security requirements

- Key agreement: If two honest parties locally generate keys associated with each other then the keys are identical.
- Key secrecy: The key must be unknown to an adversary.

Encryption-based protocol

[based on Needham-Schroeder-Lowe,78+95]

$\boxed{A}$ (knows B’s public encryption key EB)

$\boxed{B}$ (knows A’s public encryption key EA)

A: Choose a random k-bit N_A

$$\xrightarrow{ENC_{EB}(N_A, A, B)}$$

B: If decryption and identity Checks are ok then Choose a random k-bit N_B and send

$$\xleftarrow{ENC_{EA}(N_A, N_B, A, B)}$$

A: If identity and nonce checks are ok then output N_B and send

$$\xrightarrow{ENC_{EB}(N_B)}$$

B: If nonce check is ok then Output N_B

The protocol satisfies the requirements:

- Key agreement: If A, B locally output a key with each other, then this key must be N_B.
  (Follows from the "untamperability" of the encryption.)

- Key secrecy: The adversary only sees encryptions of the key, thus the key remains secret. (Follows from the secrecy of the encryption.)

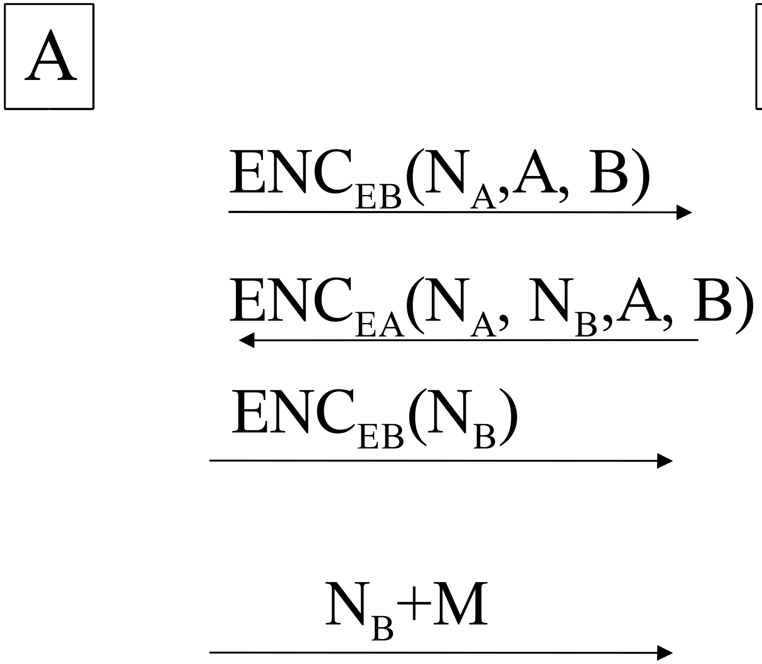
Indeed, the protocol complies with early notions of security (e.g. [Dolev-Yao83, Bellare-Rogaway93, Datta-Derek-Mitchell-Warinschi06]).

Using the key for encrypting messages

Assume that the protocol is “composed” with an encryption protocol that uses the generated key to encrypt messages. Furthermore:

-The encryption protocol is one-time-pad

-The message is either “buy” or “sell”:



A

B

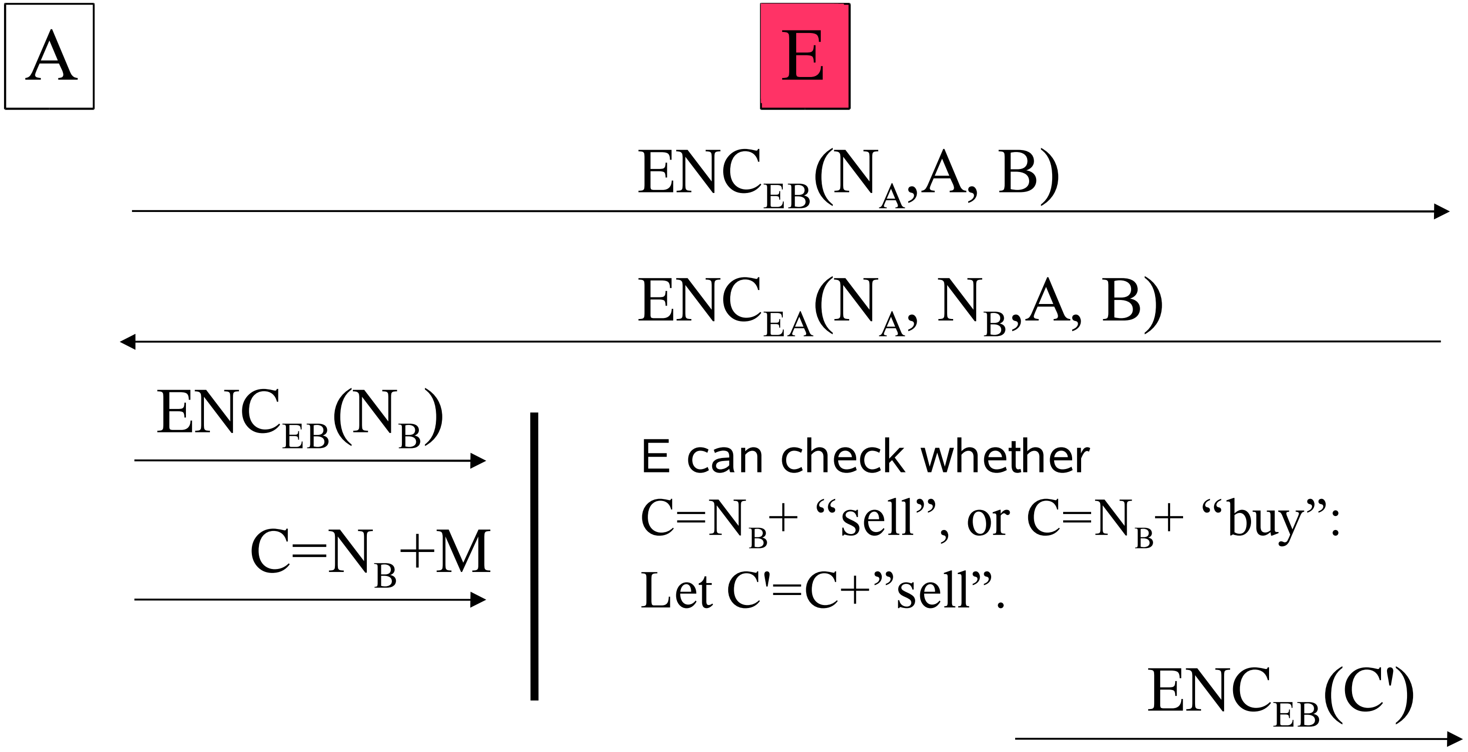
$ENC_{EB}(N_A, A, B)$

$ENC_{EA}(N_A, N_B, A, B)$

$ENC_{EB}(N_B)$

$N_B + M$

An attack against the composed protocol:



A

E

B

$ENC_{EB}(N_A, A, B)$

$ENC_{EA}(N_A, N_B, A, B)$

$ENC_{EB}(N_B)$

$C=N_B+M$

E can check whether
$C=N_B+$ “sell”, or $C=N_B+$ “buy”:
Let $C'=C+$”sell”.

$ENC_{EB}(C')$

Note: If M= ”sell” then $C'=(N_B+$”sell”$)+$”sell”$=N_B$. Else $C' \mathrel{!}= N_B$.
Thus, B accepts the exchange if and only if M= “sell”.

The problem: The adversary uses B as an “oracle” for whether it has the right key.

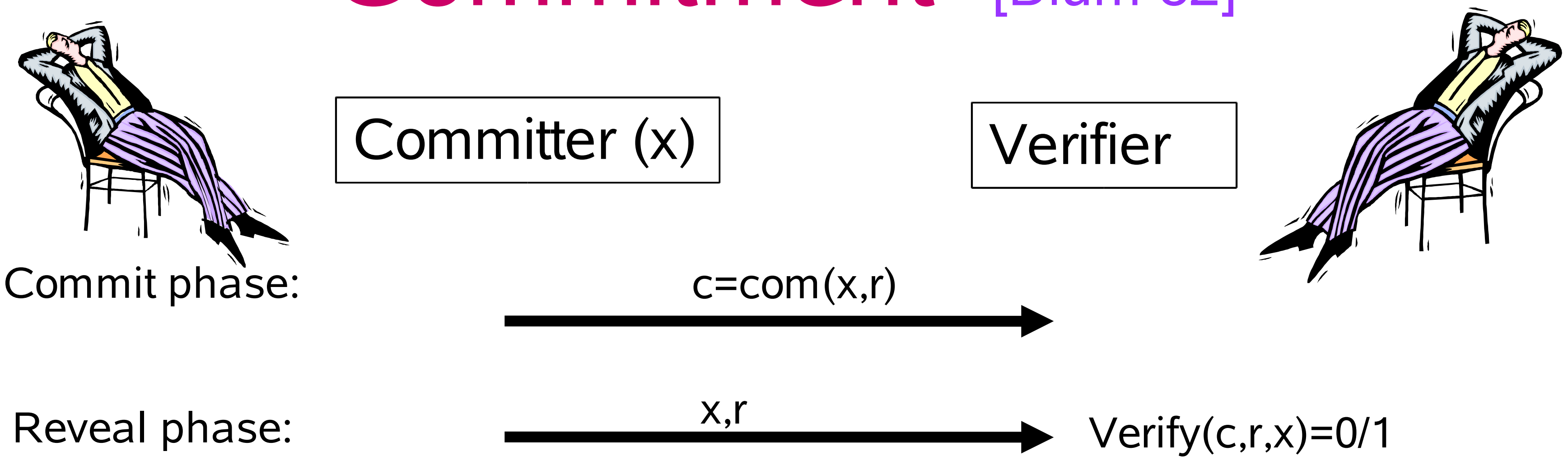
But the weakness comes to play only in conjunction with another protocol (which gives the adversary two possible candidates for the key…)

Consequently, need to explicitly incorporate the encryption protocol in the analysis of the key exchange protocol…

Insufficiency of stand-alone security

3rd example: Malleability of commitment

Commitment [Blum 82]

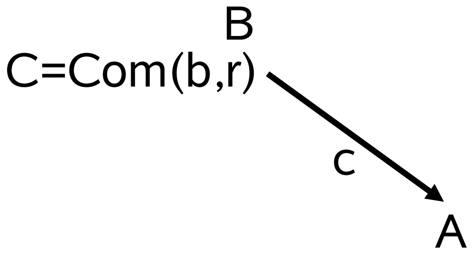


Traditional security properties:

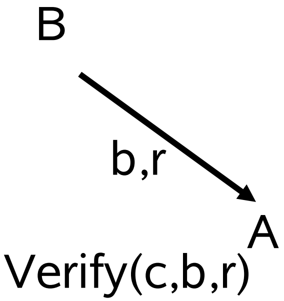
- Binding: The committer can open c to only a single value (i.e., cannot find c,r,r',x!=x' such that Verify(c,r,x)=Verify(c,r',x')=1)
- Secrecy: c gives the verifier no information on x (i.e., for any x,x', com(x,r) ~ com(x,r'))

An auction protocol using commitments:

Phase 1:
Each bidder publishes
a commitment to its bid, b.

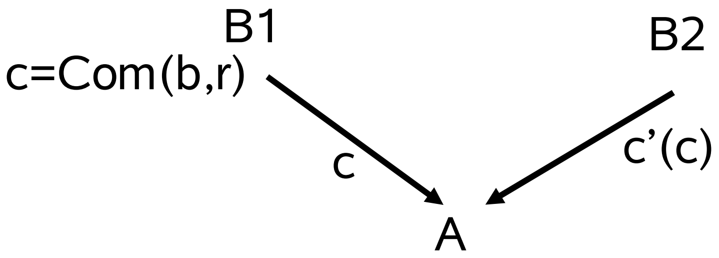

B
C=Com(b,r)
c
A

Phase 2:
Bidders open
their commitments.


B
b,r
A
Verify(c,b,r)

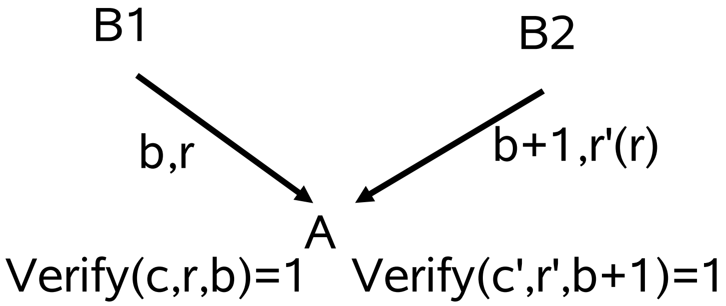
An attack on the auction protocol:

Phase 1:
Each bidder publishes
a commitment to its bid.



B1
B2
c=Com(b,r)
c
c’(c)
A

Phase 2:
Bidders open
their commitments.



B1
B2
b,r
b+1,r'(r)
A
Verify(c,r,b)=1
Verify(c',r',b+1)=1

The problem: The stand-alone definition does not guarantee that the committed values in different instances are independent from each other.

This is a new security concern, that does not exist in the stand-alone model…

Non-malleable commitments

[Dolev-Dwork-Naor 91]

Guarantee “input independence” for commitments in the case where *two* instances of the *same* commitment protocol run concurrently.

Non-malleable commitments

[Dolev-Dwork-Naor 91]

Guarantee “input independence” for commitments in the case where *two* instances of the *same* commitment protocol run concurrently.

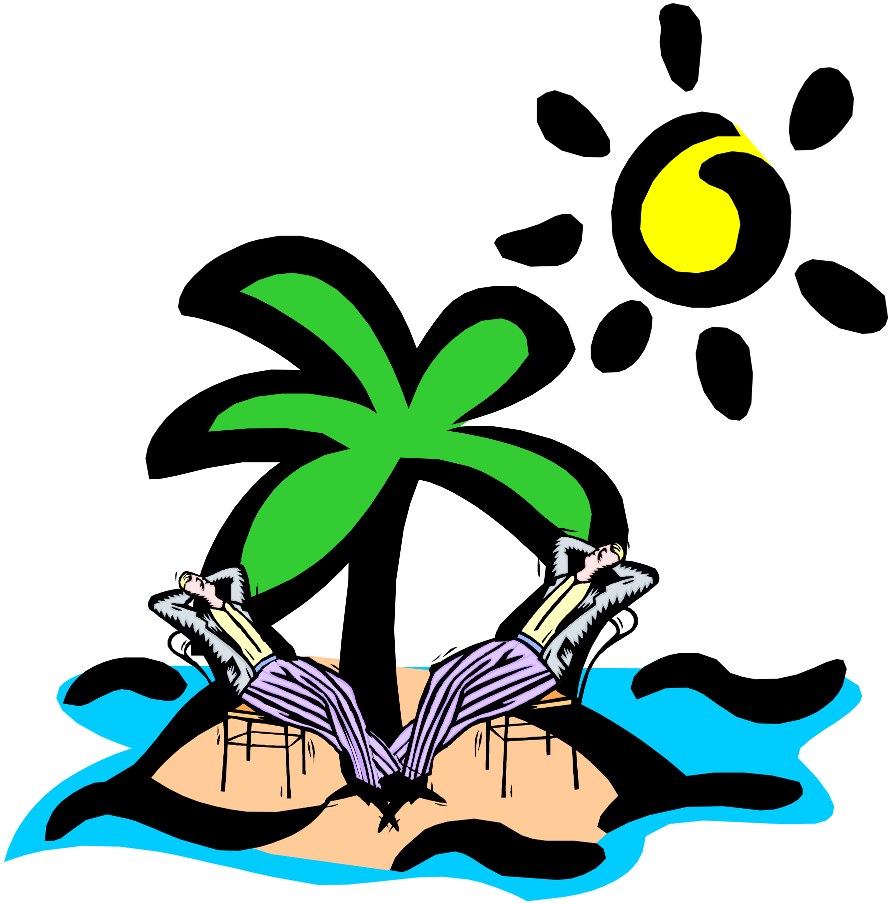
What about multiple instances? Different protocols?

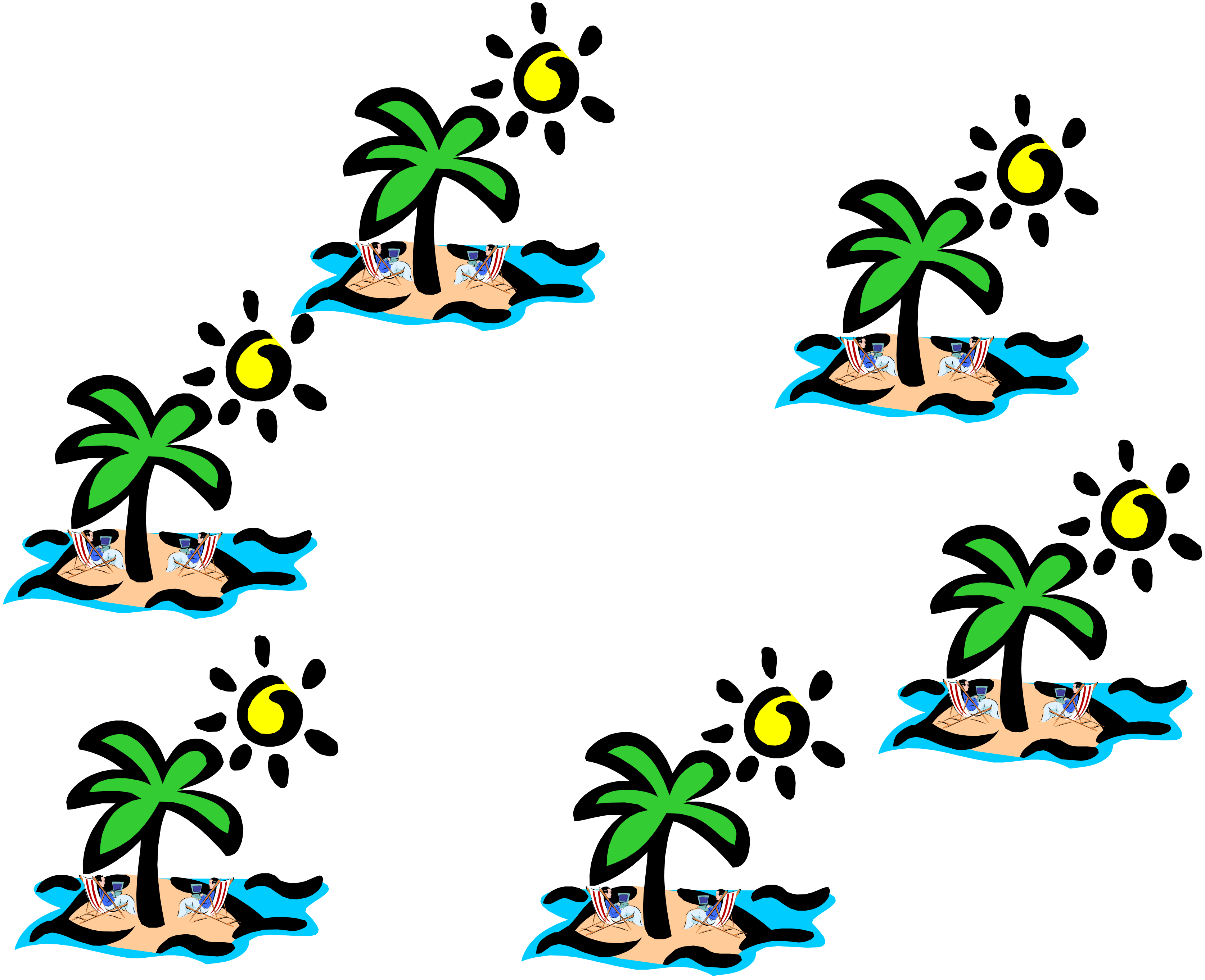
Seems hopeless:

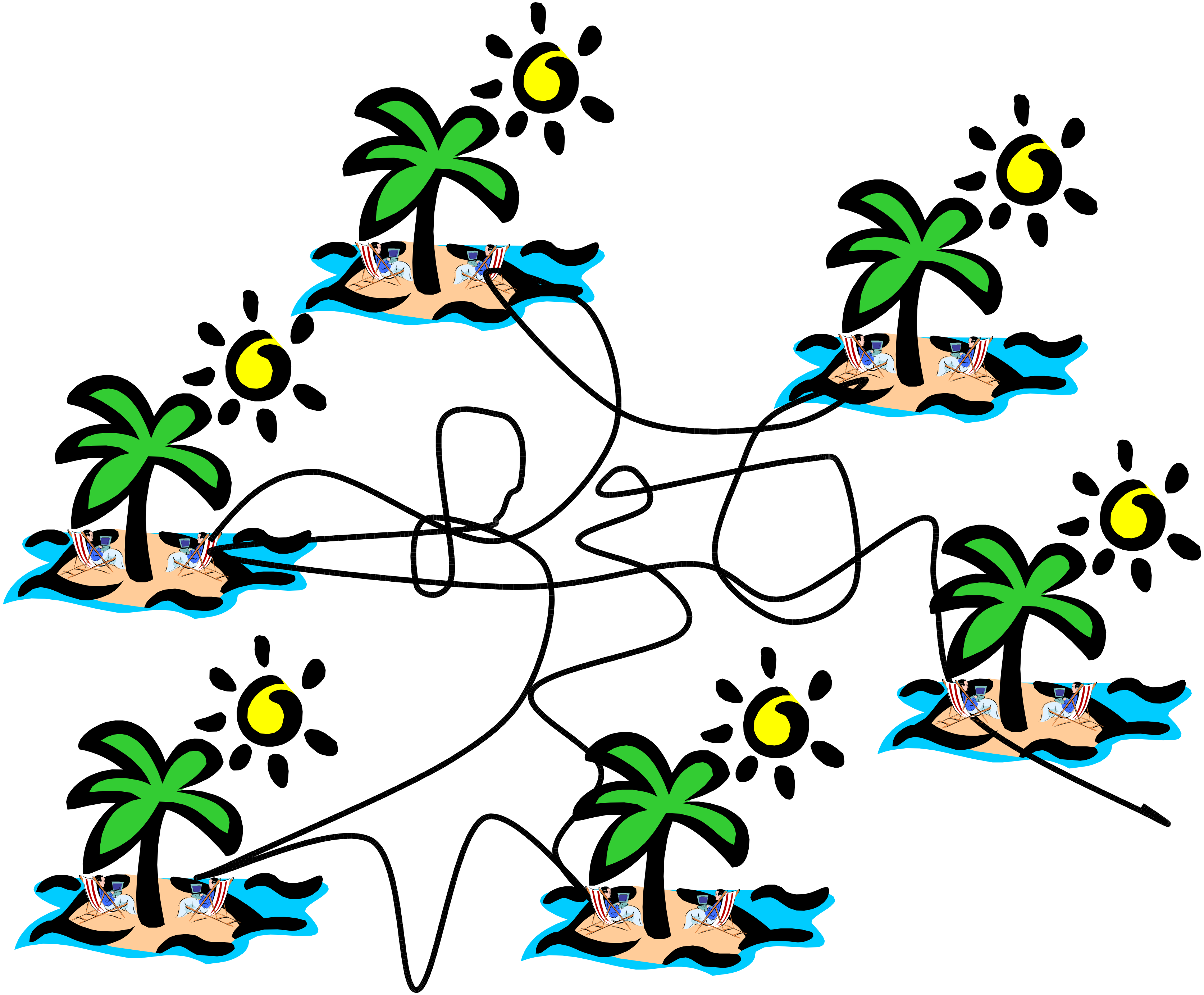
- Given a commitment protocol C, define the protocol C':
  - To commit to x, run C on x-1.
- Now, all the attacker has to do is to claim it uses C' and copy the commitment and de-commitment messages…

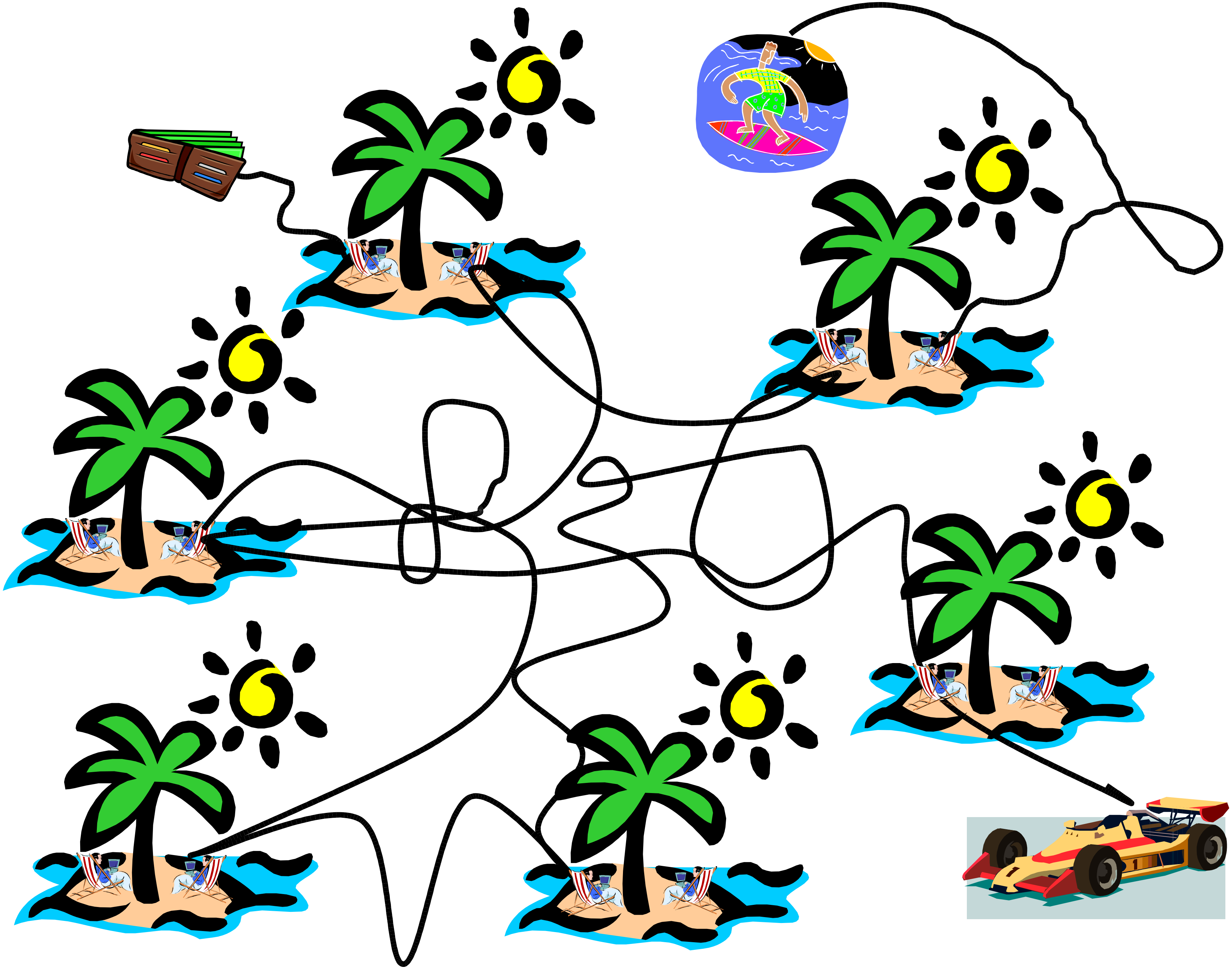
Insufficiency of stand-alone security: Other examples

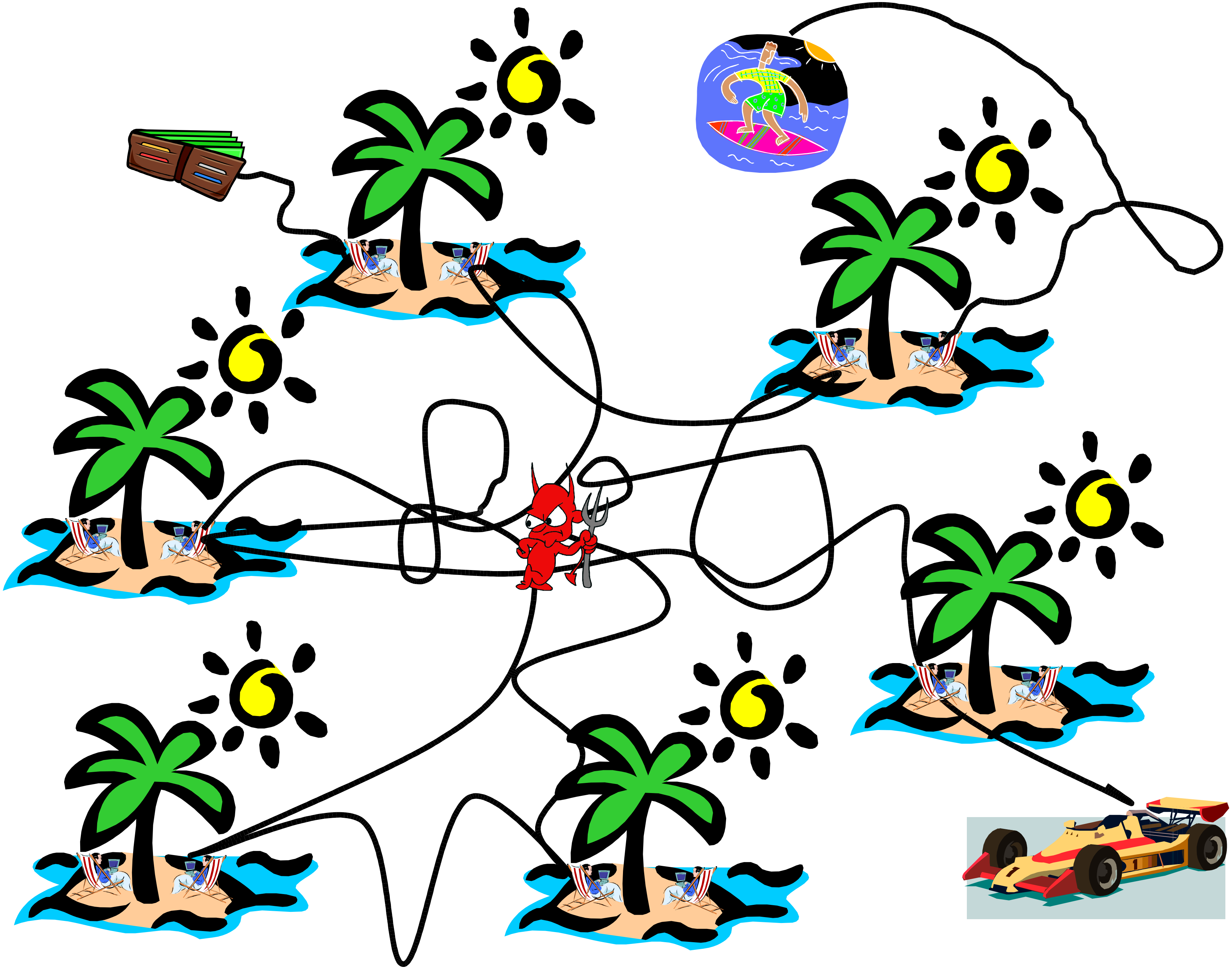
- Zero-Knowledge protocols:
  - The original (stand-alone) notion does not guarantee ZK for even *two* concurrent copies [Goldreich-Krawczyk88].
  - Obtaining ZK when the number of concurrent copies is unbounded becomes even harder [C-Kilian-Petrank-Rosen01].
- Byzantine Agreement:
  - Obtaining “concurrent BA” is impossible for t>n/3, ever with set-up [Lindell-Lysyanskaya-Rabin02].
- …











How to guarantee security in complex protocol environments?

- Traditionally: Keep writing more and more sophisticated requirements, that capture more and more scenarios…
  - Ever more complex
  - No guarantee that "we got it all".

An alternative approach:

- Prove security of a protocol as stand-alone (single execution, no other parties).
  - Use a general **secure composition theorem** to deduce security in arbitrary execution environments.

Pre-requisites for a viable “secure composition” approach

Need:

- A general framework for representing security concerns and requirements from protocols

- A general composition operation that:
  - Captures realistic situations in multi-protocol systems
  - Preserves security

Developing a general framework for representing security of protocols

[Goldwasser-Levin 90], [Micali-Rogaway 91], [Beaver 91], [Mitchel-Mitchell-Schedrov 98], [Hirt-Maurer 00], [Dodis-Micali 00], [Backes-Pfitzmann-Waidner 93,00,01,04], [Canetti 92,00,01,05]…

Main issues:

- Expressibility: How to allow expressing realistic situations and concerns.
- General composability: How to formulate a composition operation that represents how protocols "compose" in reality.
- Security preservation: How to prove that such a composition operation preserves security.

Universally Composable (UC) Security

- A framework that allows expressing any set of concerns and requirements for any crypto task:

  (e.g. authenticity, secrecy, anonymity, privacy, correctness,unpredictability, fairness, public verifiability, coercion-freeness, termination, availability…)
- A composition operation ("universal composition") that allows expressing practically any way in which protocols interact and compose.
- A way to assert security that's preserved under universal composition.

The basic paradigm

[Goldreich-Micali-Wigderson87]

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

Intuitively very attractive.

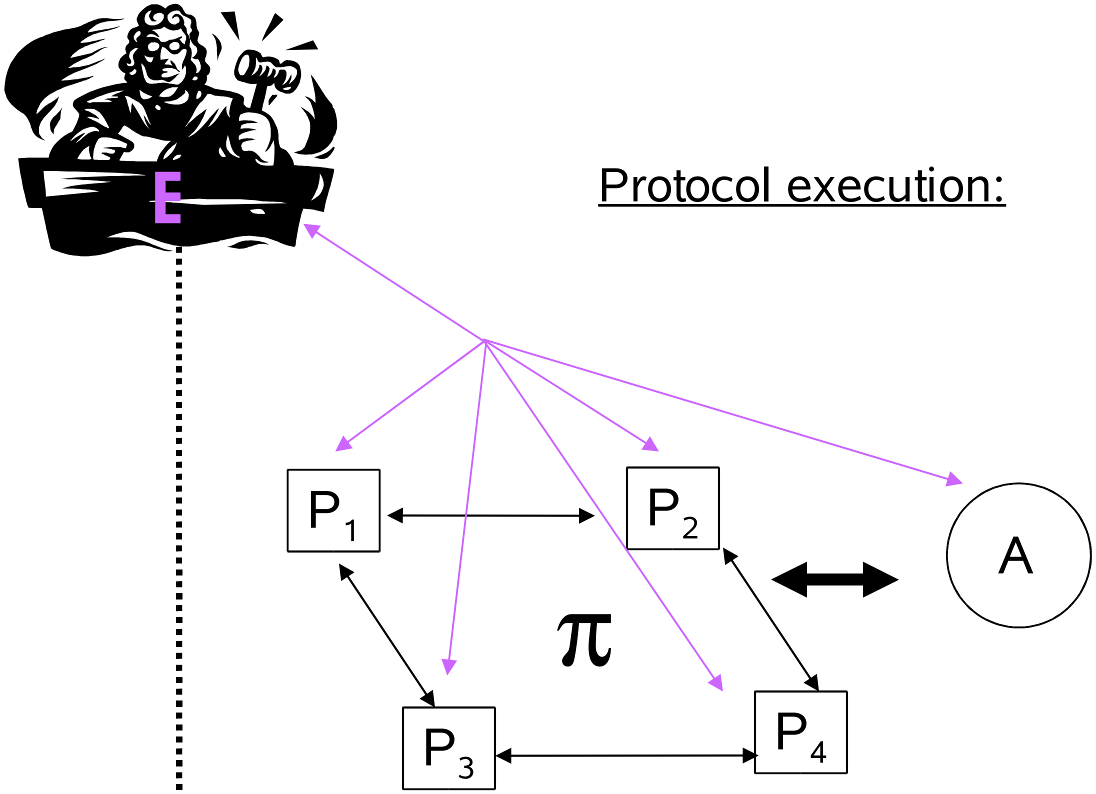
But, how to formalize?

UC security in a nutshell

Will present in three steps:

- Formalize the process of protocol execution in presence of an adversary
- Formalize the “ideal process” for realizing the functionality
- Formalize the notion of “a protocol emulates the ideal process for realizing a functionality.”

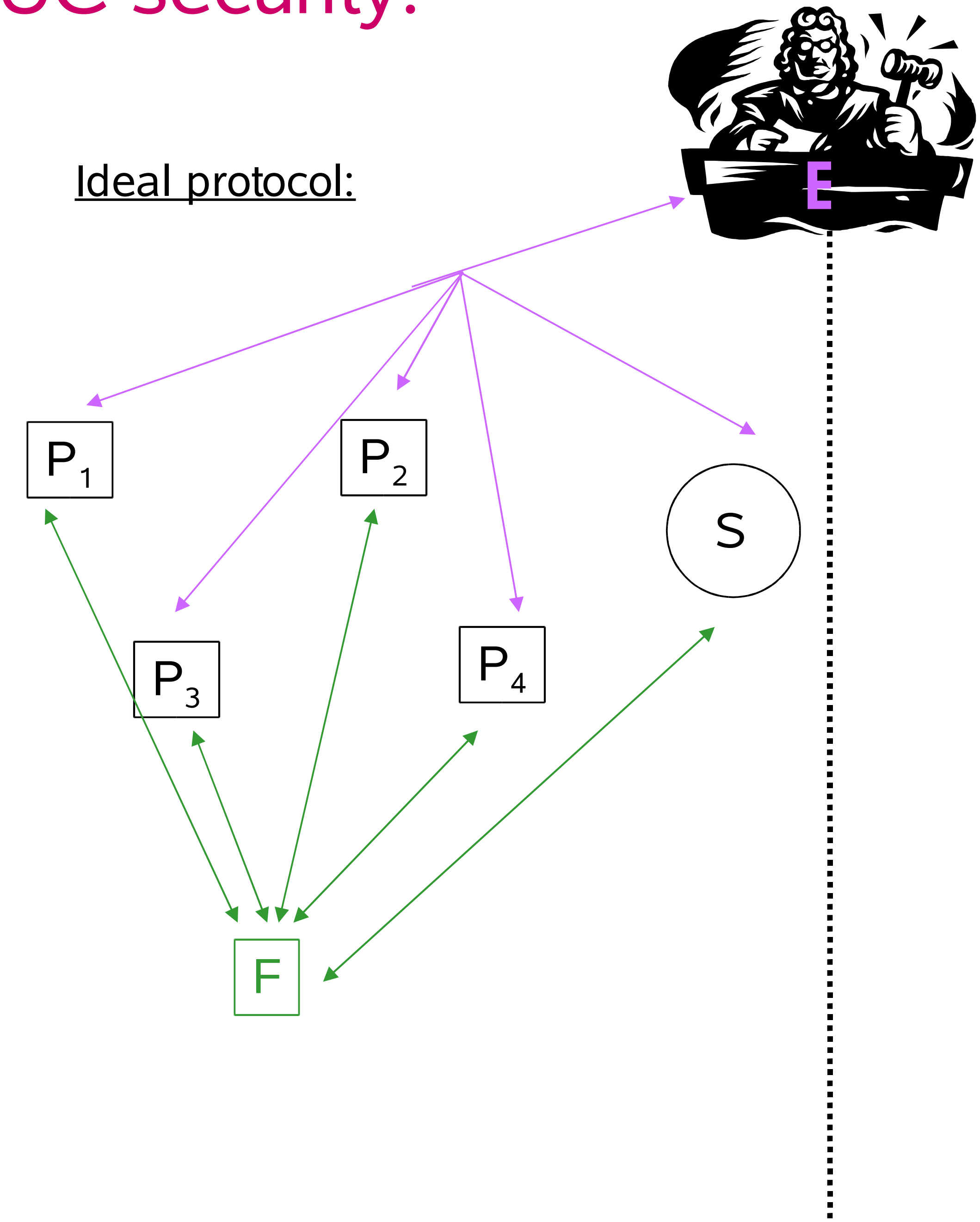
UC security:



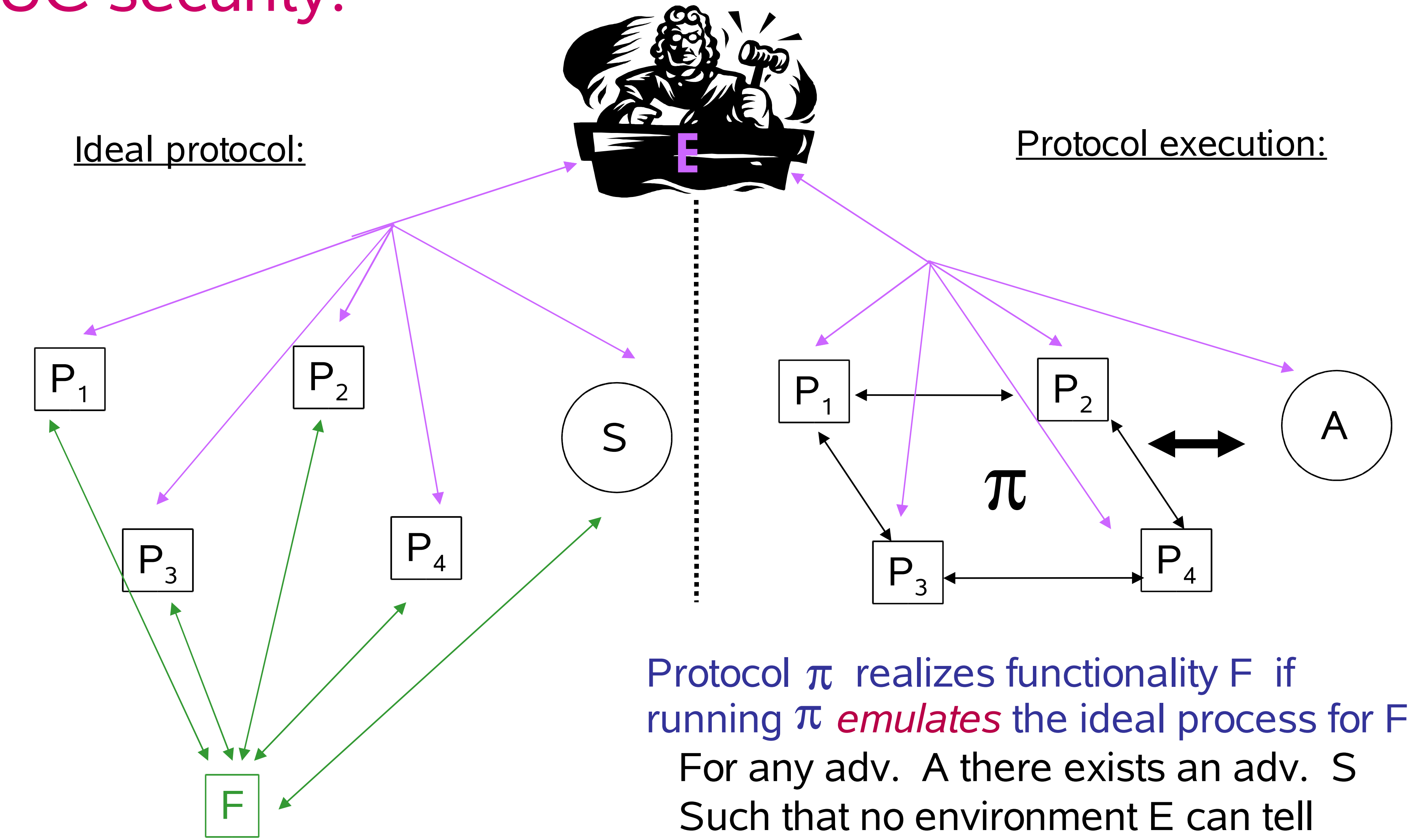
Protocol execution:

UC security:

Ideal protocol:



UC security:



Ideal protocol:

Protocol execution:

Protocol π realizes functionality F if running π *emulates* the ideal process for F:

For any adv. A there exists an adv. S Such that no environment E can tell whether it's interacting with:

- A run of π with A
- An ideal run with F and S

Implications of the definition

Correctness: In the ideal process the parties get the “correct” outputs, based on the inputs of all parties. Consequently, the same must happen in the protocol execution (or else Z will tell the difference).

Secrecy: In the ideal process the adversary learns nothing other than the outputs of bad parties. Consequently, the same must happen in the protocol execution.

Fairness, Availability, etc.: Argued in a similar way.

Example:
The Ideal Millionaires Functionality

1. Receive (x) from party A
2. Receive (y) from party B
3. Set b=x>y. Send (b) to A and B, and halt.

Each party is assured that:
- Its own output is correct, based on the other's input
- The input contribted by the other is independent of its own
- Its own input is secret, except for the function value

Example:
The Ideal Key Exchange Functionality

1. Receive (sid, B) from party A
2. Receive (sid, A) from party B
3. Choose a random key k and output (sid,A,B,k) to A and B.

The parties are assured that:

- They obtain the same key
- The key is random and known only to them.

(In fact, this is too ideal. Need to address corrupted peers, non-blocking, etc…)

Example:
The Ideal Commitment Functionality

1. Upon receiving (“commit”,C,V,x) from C, record x, and send (C, “receipt”) to V.
2. Upon receiving (“open”) from C, send (C,x) to V and halt.

Note:

- C is assured that V learns nothing about x prior to opening.
- V is assured that the value x it received in step 2 was fixed in step 1. Furthermore, x was chosen based only on what's known to V at the time.

The big gain:
Security-preserving protocol composition

The composition operation:

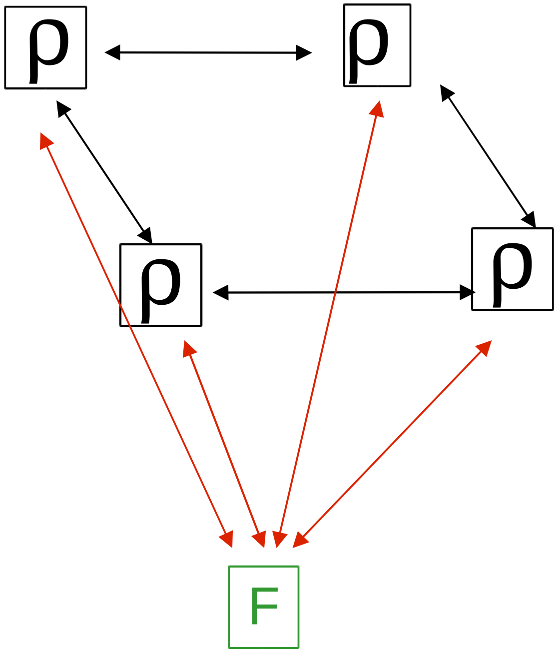
Start with

- Protocol ρ that uses ideal calls to functionality F
- Protocol π that securely realizes F

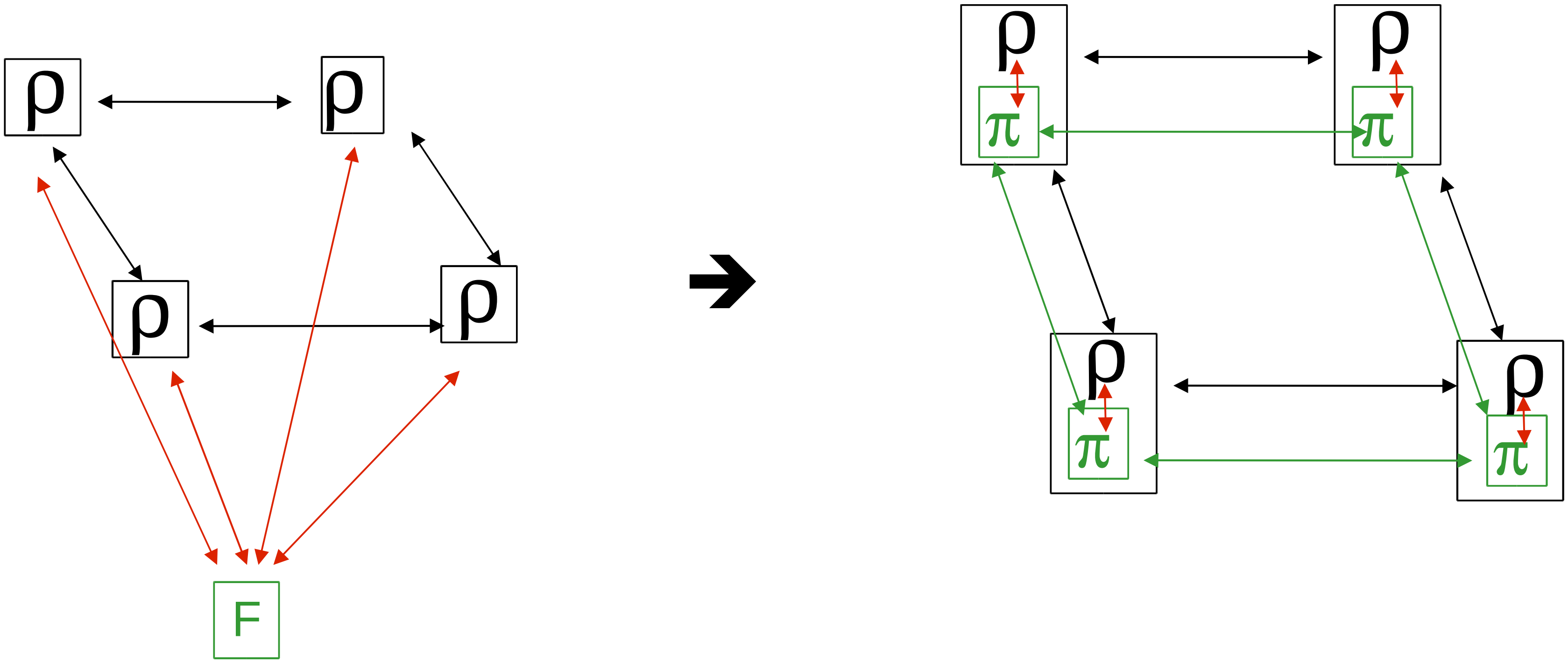
Construct the composed protocol ρ^{π}:

- Each call to F is replaced with an invocation of π.
- Each value returned from π is treated as coming from F.

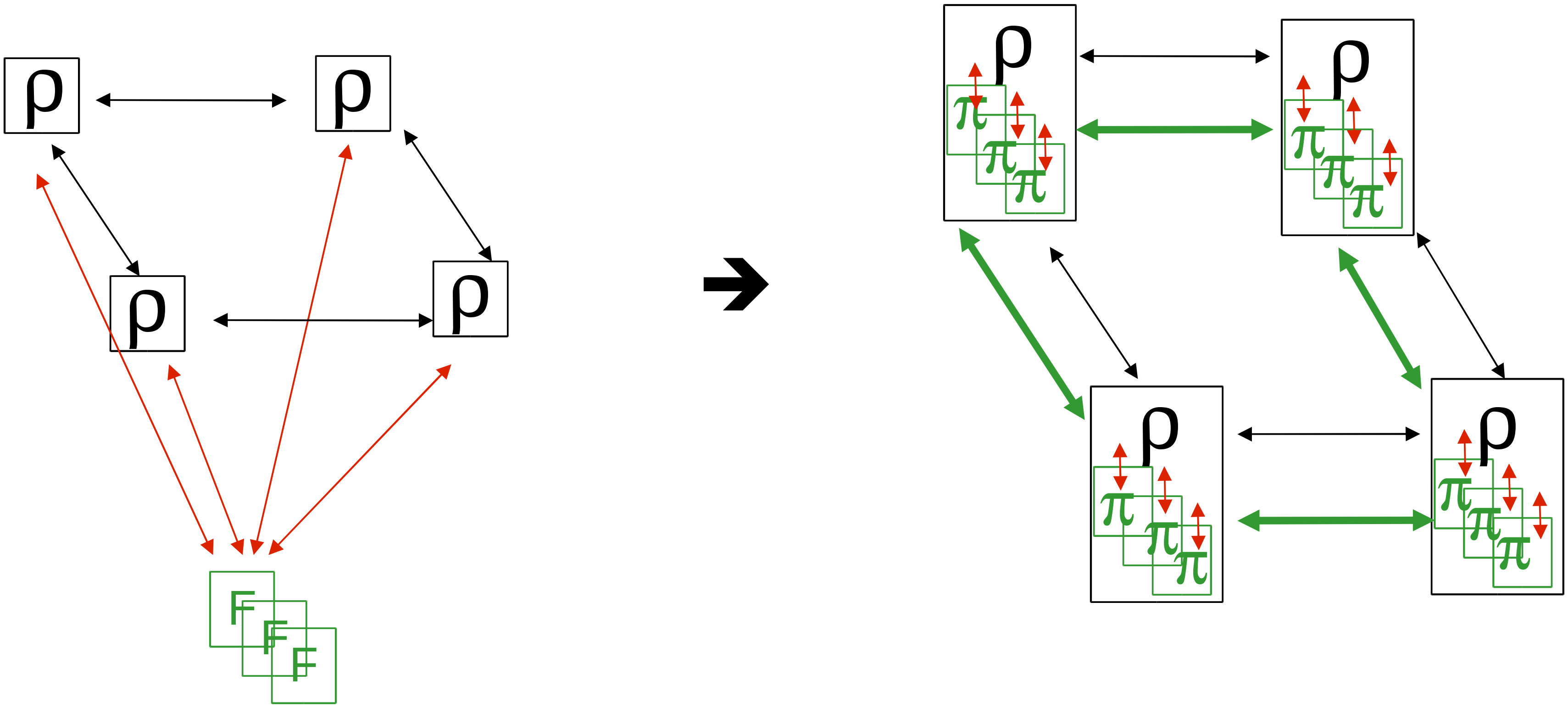
The composition operation (single call to F)



The composition operation
(single call to F)



The composition operation (multiple calls to F)



The universal composition theorem:

Protocol ρ^{π} emulates protocol ρ.

(That is, for any adversary A there exists an adversary S such that no E can tell whether it is interacting with (ρ^{π},A) or with (ρ,S).)

Corollary:

If ρ realizes functionality G then so does ρ^{π}.

The universality of universal composition

Captures all common ways to combine protocols:

- Subroutine calls
- Sequential, parallel, concurrent, executions
- Executions by same party, by unrelated parties
- Executions on same/related inputs, on unrelated inputs
- Unbounded number of executions
- Dynamic and adversarial code generation (“chosen protocol attacks”)

Two benefits of the UC theorem

- Security in complex environments:
  - Guarantee security when the protocol is running alongsi other (potentially unknown) protocols.
- Modular design and analysis of systems:
  - De-compose a complex system into small protocols.
  - Analyze the security of each protocol separately (as stand-alone).
  - Deduce the security of the composite system.

Questions:

- Is UC security achievable?
  - Are existing protocols enough?
  - Can we design new protocols that suffice?
- Can we relax the definition and still guarantee both meaningful security and composability?

Highlights of current answers

- For “secure communication primitives” (authentication, encryption, digital signatures, key-exchange, etc.):
  - Many known protocols are UC-secure (e.g., IKE/SIGMA, TLS, NSL,…) [Backes-Pfitzmann-Waidner01,03,04,C-Krawczyk02…]
  - Some notions are equivalent to traditional ones (e.g., digital signatures, CCA-secure encryption) [C01,04,C-Krawczyk-Nielsen03,Hofheinz-QuadeMuller-Unruh04,….]

Highlights of current answers

- For “secure communication primitives” (authentication, encryption, digital signatures, key-exchange, etc.):
  - Many known protocols are UC-secure (e.g., IKE/SIGMA, TLS, NSL,…) [Backes-Pfitzmann-Waidner01,03,04,C-Krawczyk02…]
  - Some notions are equivalent to traditional ones (e.g., digital signatures, CCA-secure encryption) [C01,04,C-Krawczyk-Nielsen03,Hofheinz-QuadeMuller-Unruh04,….]

- For “general multiparty computation”:
  - With honest majority, known protocols are UC-secure. [BenOr-GoldWasser-Wigderson88,BenOr-Rabin89,C-Feige-Goldreich-Naor96,C01]

Highlights of current answers

- For general multiparty computation with honest minority:
  - Many “traditional” protocols don't work
  - Many tasks are impossible to obtain “from scratch” [C-Fischlin01,C-Kushilevitz-Lindell03, Datta-Derek-Mitchell+06…]
  - Can regain general feasibility with set-up assumptions (e.g. common reference string, enhanced PKI, timing) [C-Lindell-Ostrovsky-Sahai02, Damgard-Nielsen02,Barak-C-Nielsen-Pass04,Kalai-Lindell-Prabhakaran05,C-Dodis-Pass-Walfish07,…]
  - Can regain feasibility “from scratch” with weaker notions (but have to give up on either security or composability) [Prabhakaran-Sahai04,Barak-Sahai05,Malkin-Moriarty-Yakovenko06, Micali-Pass-Rosen06,…]

Two applications in more detail:

- Modular design and analysis of key exchange and secure communication session protocols

- Computationally sound formal and automated analysis of protocols

A small problem…

- The UC operation/theorem applies only to instances that dont share any local state.
- In contrast, in reality many protocols often share the same state (e.g., same PK/SK pair for many key exchanges).
- We know that sharing secret state is dangerous…
- How to argue about such cases?

Universal composition with joint state (JUC)

[C-Rabin03]

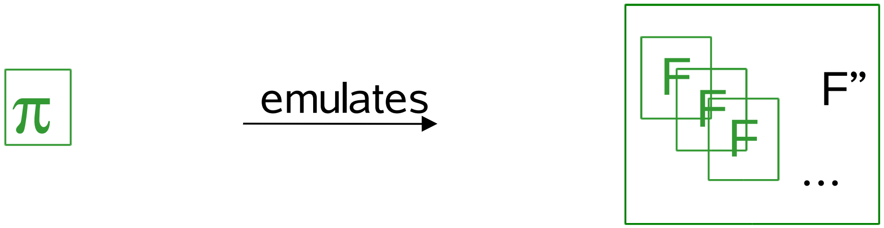
Provides a general design methodology such that:

- Protocols that comply can be composed securely with joint state
- Known protocols comply

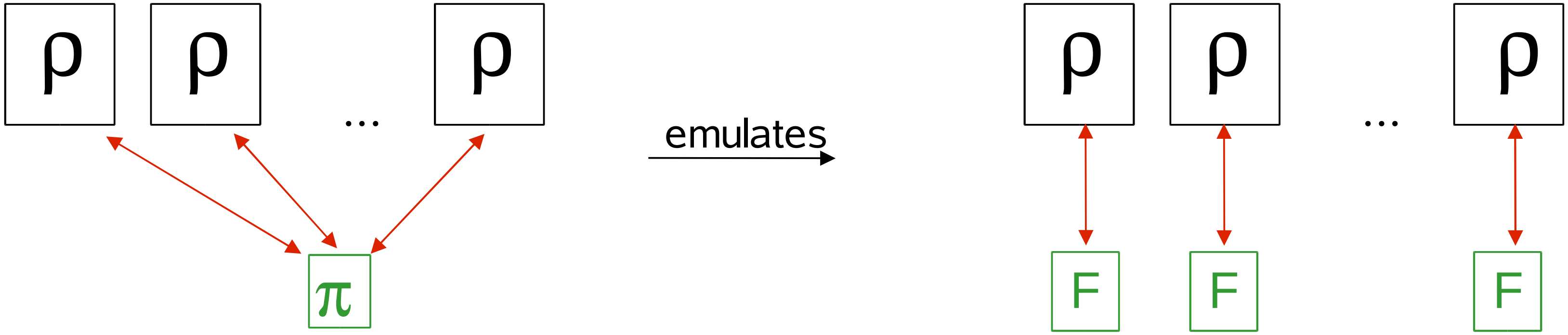
Universal composition with joint state (JUC)

[C-Rabin03]

If



then:



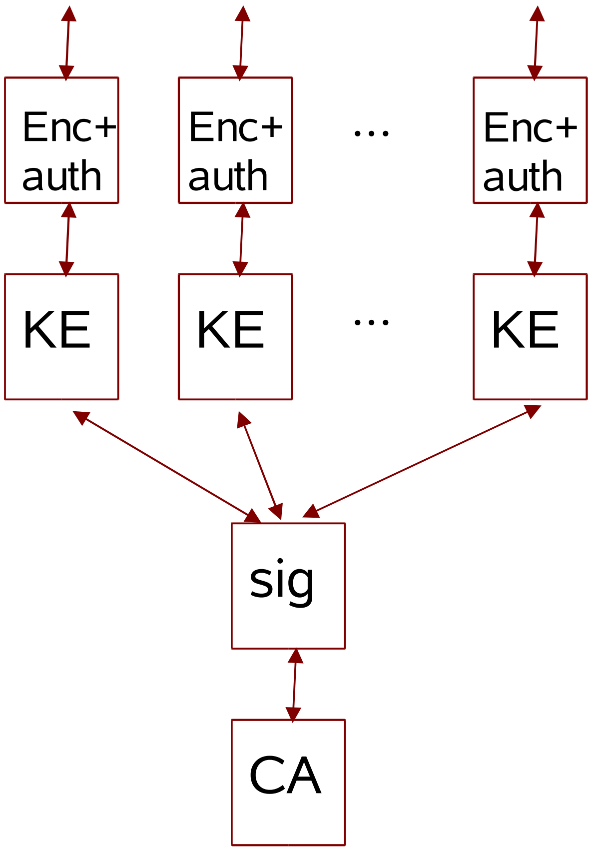
Modular analysis of secure channels

Many components:

- Registration (CA, authentication servers)
- Key exchange
- Data encryption
- Date authentication
- Replay protection

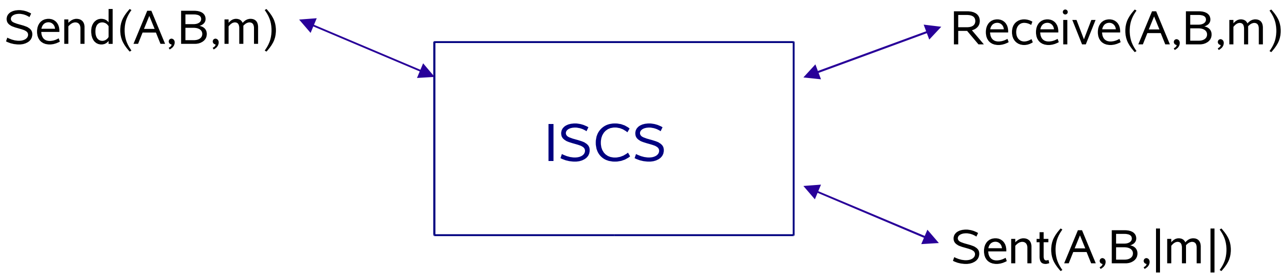
How to analyze?

The general structure of signature-based protocols (common to IPSEC,SSL/TLS, most others…)



Step 0: Set the goal

The Ideal Secure Communication Session functionality:



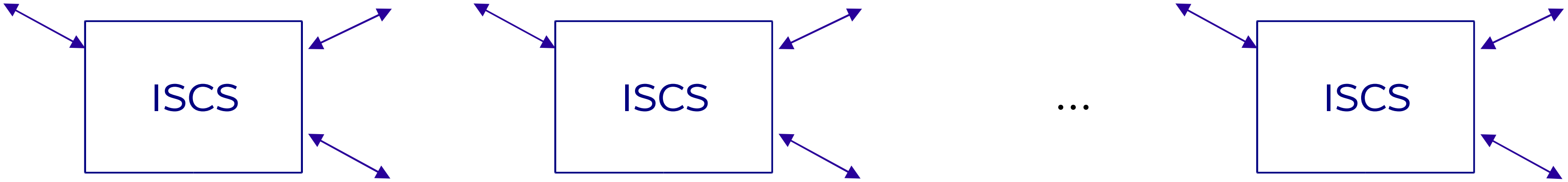
Send(A,B,m)

ISCS

Receive(A,B,m)

Sent(A,B,|m|)

Want to emulate multiple independent instances of ISCS:



ISCS

ISCS

…

ISCS

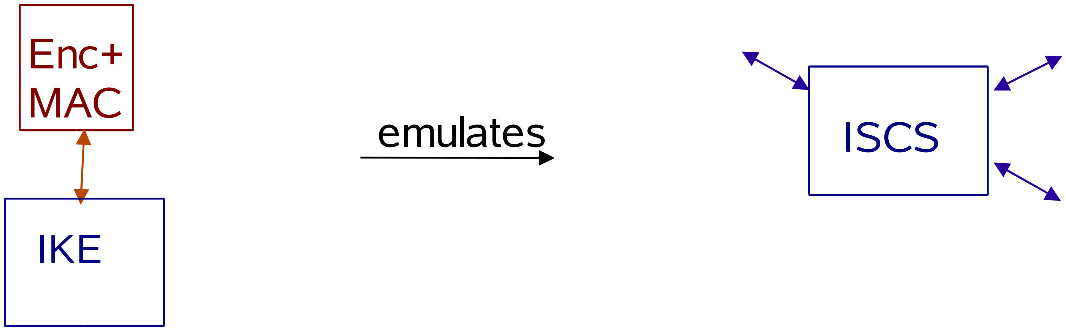
Step 1: From key exchange to secure sessions

Recall: the Ideal key exchange funtionality

Emulating ISCS given IKE:

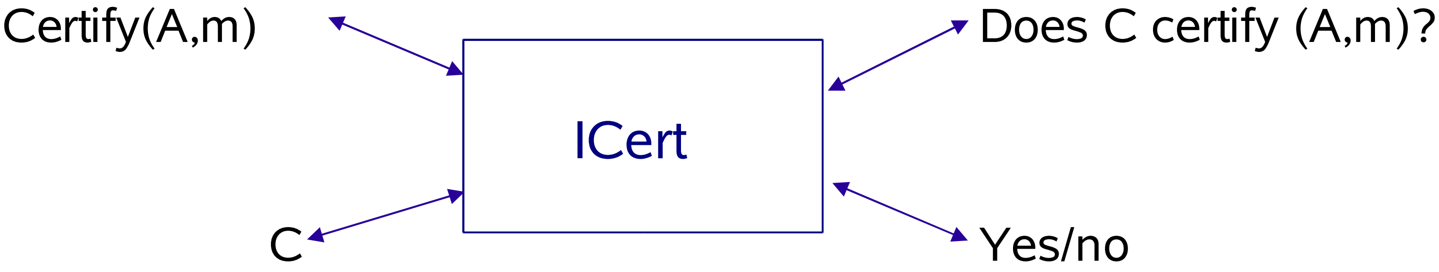
Encrypt and MAC each message with keys derived from k.

Need to demonstrate:



Step 2: From ideal certification to key exchange

Ideal certification functionality



Certify(A,m)

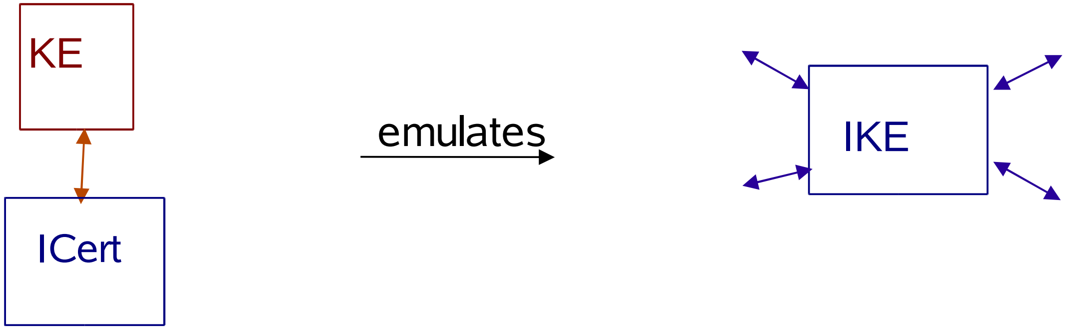
Does C certify (A,m)?

ICert

C

Yes/no

Emulating IKE given ICert: Any signature-based key-exchange protocol. Need to demonstrate:



KE

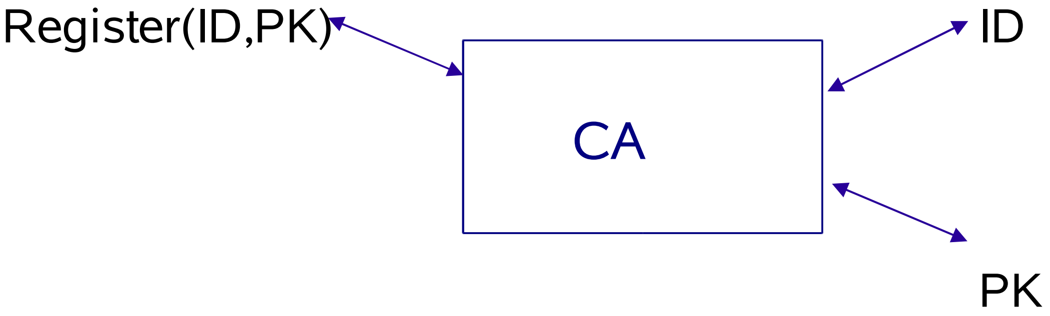
ICert

emulates

IKE

Step 3: From signatures+PKI to Multi-session ICert

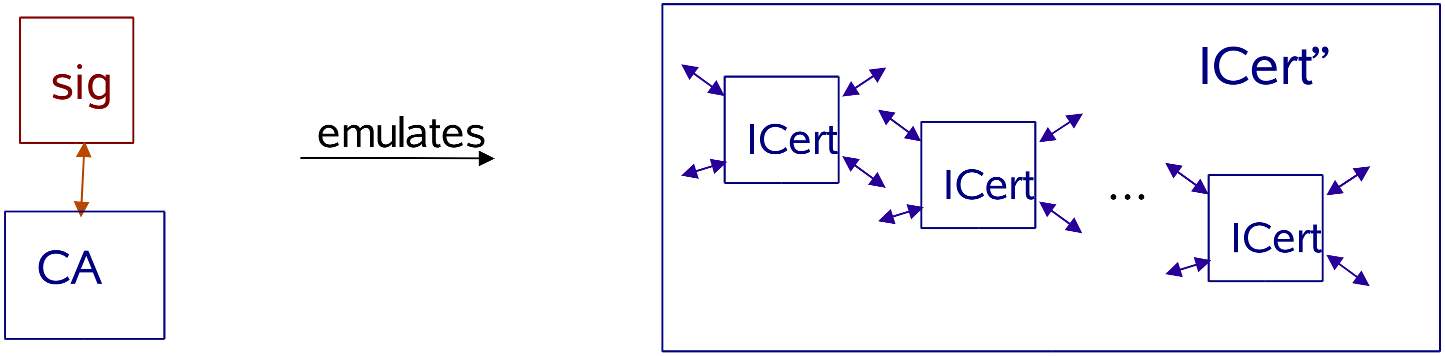
PKI functionality:



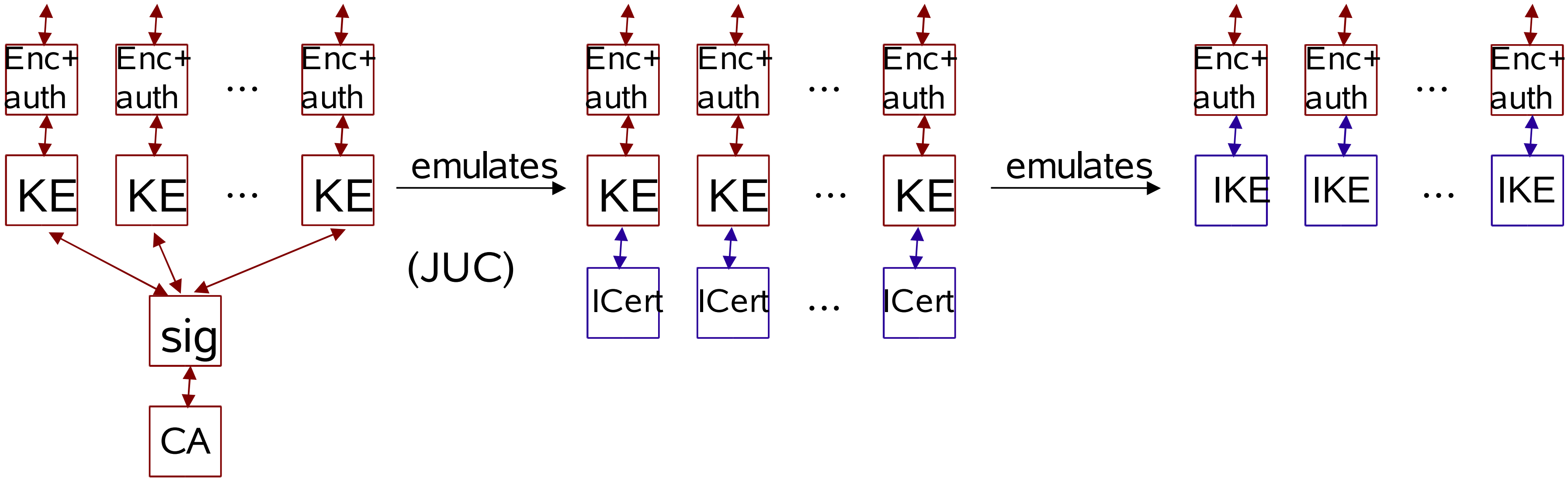
Emulating multi-session ICert given PKI:

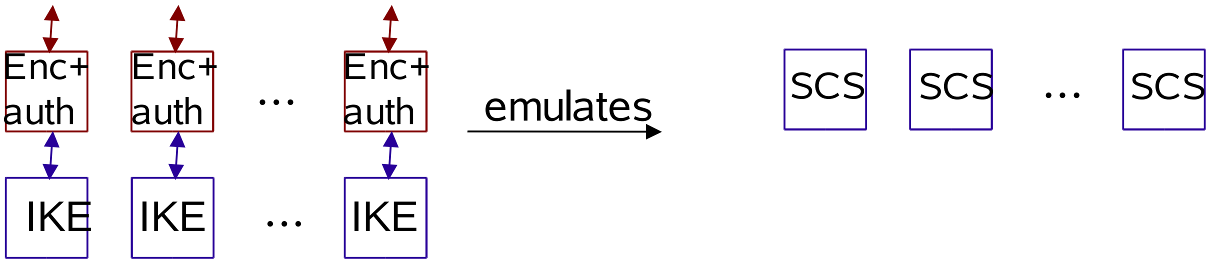
1. Register PK of a signature scheme
2. When asked to certify m within session sid, sign (sid,m).

Need to emonstrate:



Step 4: Putting things together





Reflections

- Analyzed each component separately
- Analysis of protocol only had to deal with a single session

Still:

- We could assert security for the entire system
- Security guarantees hold within any external context and for any application.

Using formal methods for security analysis

A popular method for analyzing security of cryptographic protocols, using formal tools:

Model the cryptographic operations as symbolic operations that represent “perfect cryptography”.

A quintessential example: The [Dolev-Yao83] modeling of public-key encryption and signatures. A large body of work follows this approach.

Pros and cons of “Dolev-Yao style” symbolic modeling

Main advantage:

Analysis is much simpler. Absolute assertions, no error probabilities, no computational limitations, no asymptotics. Consequently, it is amenable to automation.

Main drawbacks:

– Lack of soundness. There is no security guarantee once the symbolic operations are replaced with real cryptographic algorithms.

– No composability. Have to analyze directly an entire system. This greatly limits automation (in fact, general undecidability results exist).

Using UC security to improve symbolic modeling

Can have symbolic modeling that is:

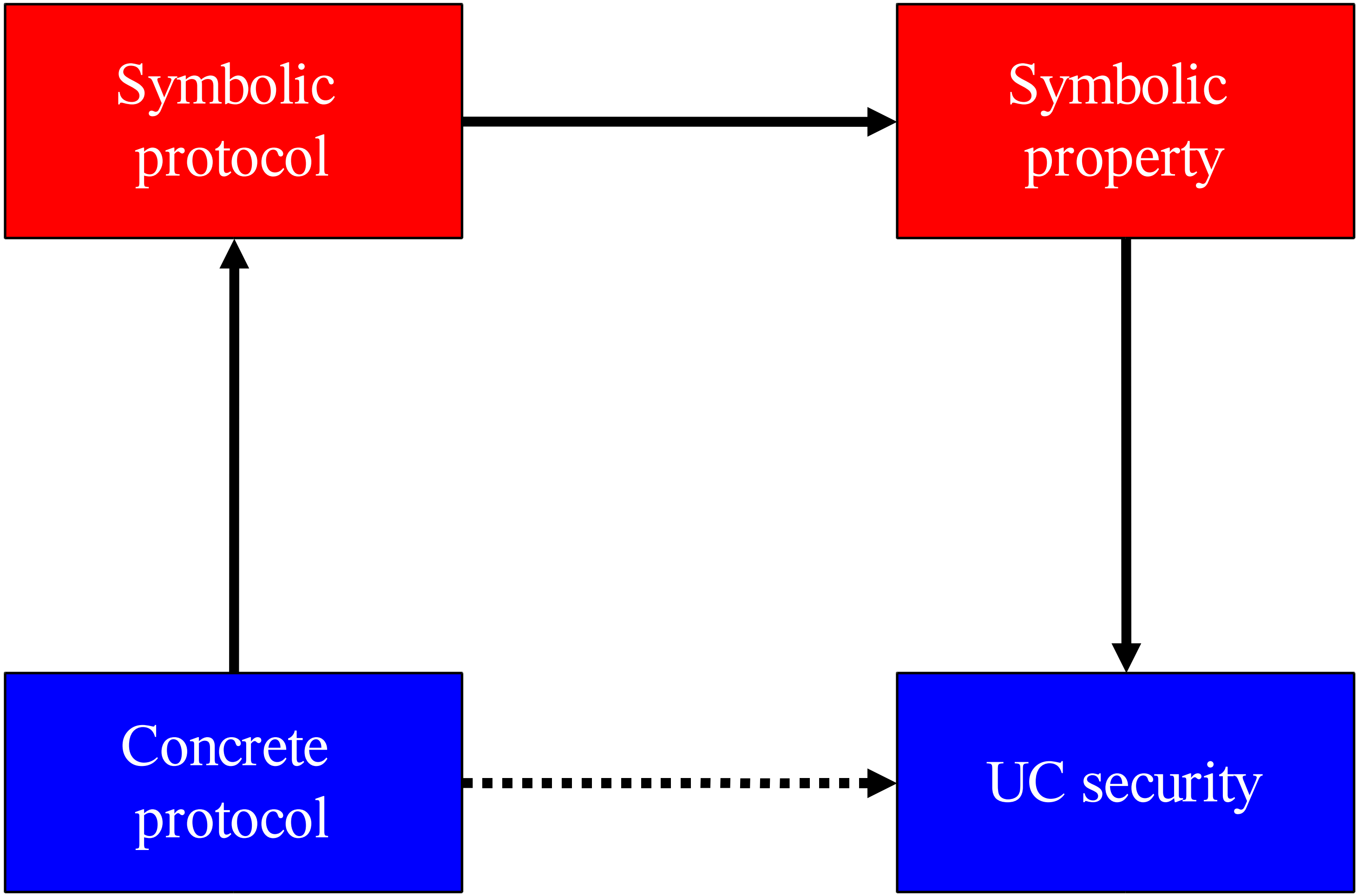
- Sound even when the symbolic operations are replaced by real algorithms.
- Avoids undecidability: Can analyze single instance protocols and deduce security of the composite system
- Automated

Main idea: Show correspondence between protocols that use symbolic crypto and protocols that use ideal functionalities in the UC framework. This allows using the universal composition theorem to obtain soundness and composability.

[Pfitzmann-Waidner00,C01,Backes-P-W-04+,C-Herzog04,Patil05]

Analysis strategy

[Abadi-Rogaway00]



Symbolic protocol

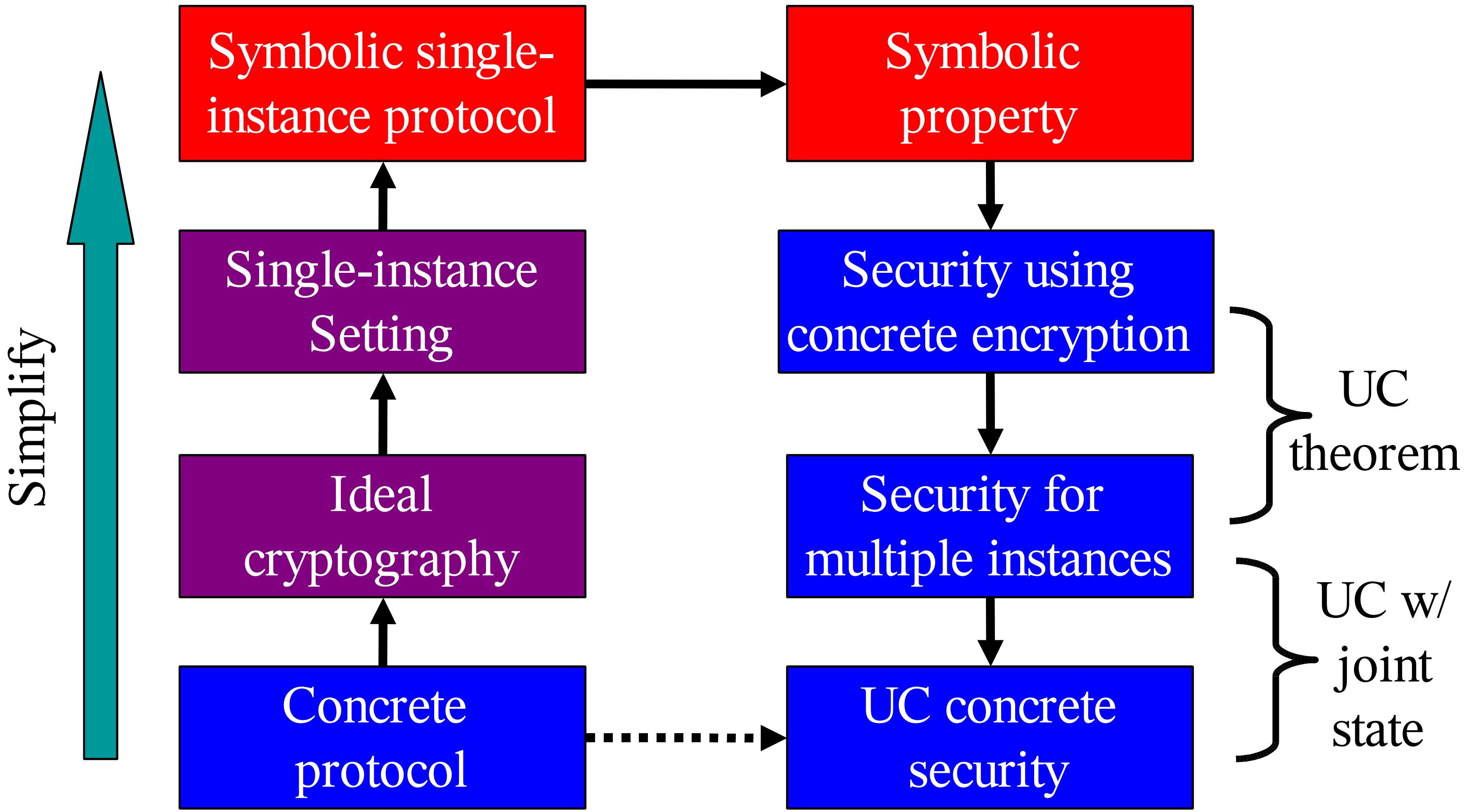
Symbolic property

Concrete protocol

UC security

Analysis strategy (expanded)

[C-Herzog04]



Simplify

Symbolic single-instance protocol

Symbolic property

Single-instance Setting

Security using concrete encryption

Ideal cryptography

Security for multiple instances

Concrete protocol

UC concrete security

UC theorem

UC w/ joint state

Further Research

Make security analysis of protocols ubiquitous

- Mechanize and automate the analysis
- Analyze real-life protocols and systems

Find better protocols that guarantee UC security

- Alternative constructions
- Alternative set-up assumptions
- Alternative ideal functionalities

Apply compositional analysis to other settings, e.g:

- Program obfuscation
- Cryptographic protocols with game-theoretic modeling (rationality, equilibria)