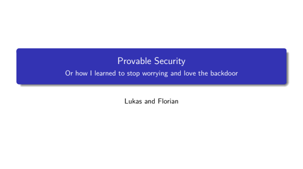
Provable Security

Or how I learned to stop worrying and love the backdoor

Lukas and Florian

2018-12-29

Provable Security

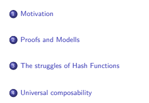

Provable Security
Or how I learned to stop worrying and love the backdoor
Lukas and Florian

- Thank you for waking up that early
- It's a great honor for us to give this talk, espessialy in the slot directly after djb and Tanja Lange
- Two points before, which might not get clear during this talk:
  - we like provable security
  - Oded Goldreich is a great cryptographer, who did amazing things for the field of cryptography

1. Motivation
2. Proofs and Modells
3. The struggles of Hash Functions
4. Universal composability

2018-12-29

Provable Security

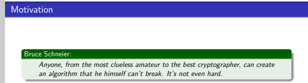

1. Motivation
2. Proofs and Modells
3. The struggles of Hash Functions
4. Universal composability

-
- Organisation of this talk:
  1. Motivation: Why do we want security **proven**
  2. Examples: What could get wrong and how to proof security of protocols
  3. 2 Examples Why modern crypto proofs are kind of weird

Motivation

Bruce Schneier:

> *Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break. It's not even hard.*



Motivation
Bruce Schneier:
Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break. It's not even hard.

Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break. It's not even hard.

- Just because you don't see the flaw in your scheme, it does not mean it's not there
- strict mathematical proofs can handle this
- But you should be aware of the boundaries

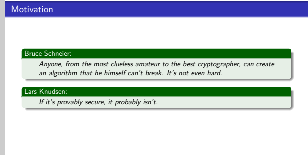
Motivation

Bruce Schneier:

> *Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break. It's not even hard.*

Lars Knudsen:

> *If it's provably secure, it probably isn't.*



Motivation

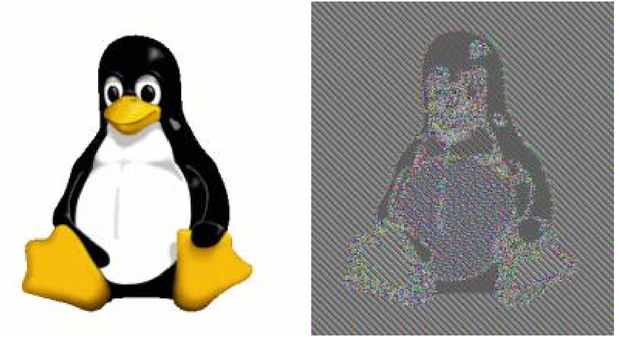
Bruce Schneier:
Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break. It's not even hard.

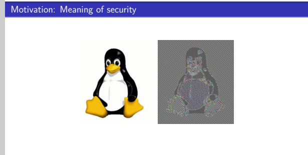
Lars Knudsen:
If it's provably secure, it probably isn't.

Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break. It's not even hard.

- Just because you don't see the flaw in your scheme, it does not mean it's not there
- strict mathematical proofs can handle this
- But you should be aware of the boundaries

Motivation: Meaning of security

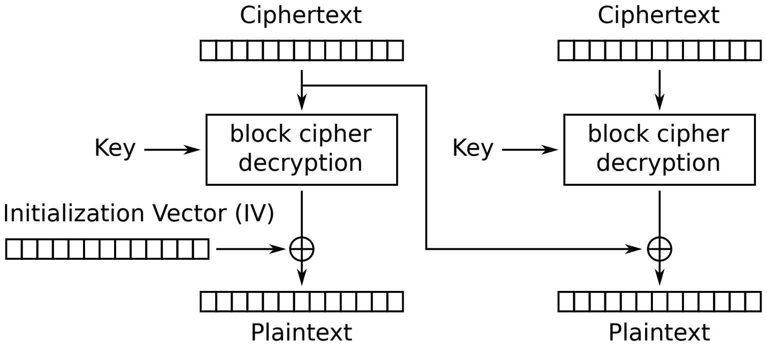


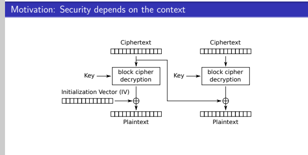


Motivation: Meaning of security

- most of you will know this example
- ECB (electronic codebook mode): mode for applying encryption defined on blocks of finite lenth to messages of arbitrary length
- each block (read: byte) is encypted in a secure way
- but deterministic
- Simply encypting each block is not a useful definition of security

Motivation: Security depends on the context




Motivation: Security depends on the context

- decryption of CBC (cipher block chaining) mode
- cipher is XORed on the output of the decryption
- deeper in the talk on TLS 1.3 by hanno

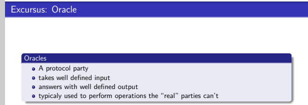
Excursus: Oracle

Oracles

- A protocol party
- takes well defined input
- answers with well defined output
- typicaly used to perform operations the “real” parties can’t

2018-12-29

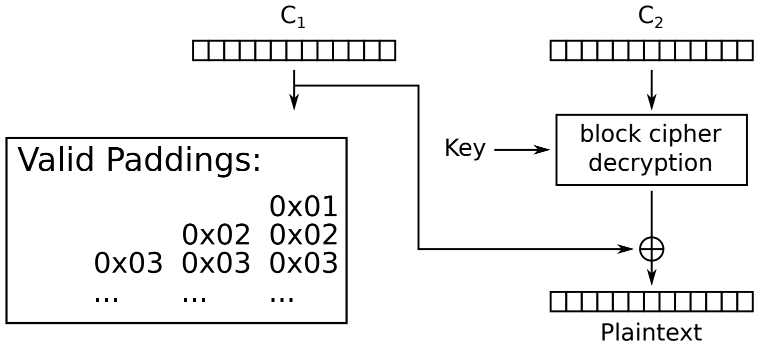
Provable Security
└Motivation
└Excursus: Oracle

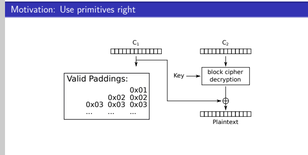

Excursus: Oracle

Oracles

- A protocol party
- takes well defined input
- answers with well defined output
- typicaly used to perform operations the “real” parties can’t

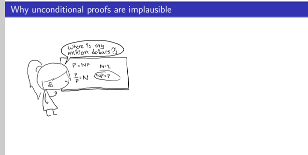
Motivation: Use primitives right




Motivation: Use primitives right

- CBC-Mode
- needs messages of specific lengths, i.e. a multiple of block size
- use padding
- excurse: Oracle
  - some magical instance
  - that takes input and
  - generates a specific Output
- Use Padding Oracle
- allows to break byte by byte
- Learn: Use your crypto right

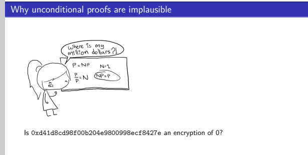
Why unconditional proofs are implausible


Why unconditional proofs are implausible

- What is P vs. NP? **Millenium Problem**
- Asume that you have build up your protocol, so let's start to prove
- breaking a cipher should be hard, which mean it should be in $\mathcal{NP} \setminus \mathcal{P}$
- PAUSE
- recognising encryptions should be hard.
- if we proof this is difficulty, we would have a Problem in $\mathcal{NP} \setminus \mathcal{P}$
- so $\mathcal{NP} \neq \mathcal{P}$

Why unconditional proofs are implausible

Is `0xd41d8cd98f00b204e9800998ecf8427e` an encryption of 0?


Why unconditional proofs are implausible

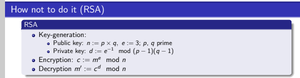
Is 0xd41d8cd98f00b204e9800998ecf8427e an encryption of 0?

- What is P vs. NP? **Millenium Problem**
- Asume that you have build up your protocol, so let's start to prove
- breaking a cipher should be hard, which mean it should be in $\mathcal{NP} \setminus \mathcal{P}$
- PAUSE
- recognising encryptions should be hard.
- if we proof this is difficulty, we would have a Problem in $\mathcal{NP} \setminus \mathcal{P}$
- so $\mathcal{NP} \neq \mathcal{P}$

How not to do it (RSA)

RSA

- Key-generation:
  - Public key: $n := p \times q,\ e := 3;\ p,\ q$ prime
  - Private key: $d := e^{-1} \mod (p-1)(q-1)$
- Encryption: $c := m^e \mod n$
- Decryption $m' := c^d \mod n$



How not to do it (RSA)

RSA

- Key-generation:
  - Public key: $n := p \times q,\ e := 3;\ p,\ q$ prime
  - Private key: $d := e^{-1} \mod (p-1)(q-1)$
- Encryption: $c := m^e \mod n$
- Decryption $m' := c^d \mod n$

How not to do it (RSA)

RSA

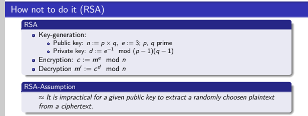
- Key-generation:
  - Public key: $n := p \times q, \ e := 3; \ p, \ q$ prime
  - Private key: $d := e^{-1} \mod (p-1)(q-1)$
- Encryption: $c := m^e \mod n$
- Decryption $m' := c^d \mod n$

RSA-Assumption

$\approx$ *It is impractical for a given public key to extract a randomly choosen plaintext from a ciphertext.*

2018-12-29

└How not to do it (RSA)



How not to do it (RSA)

RSA

- Key-generation:
  - Public key: $n := p \times q, \ e := 3; \ p, \ q$ prime
  - Private key: $d := e^{-1} \mod (p-1)(q-1)$
- Encryption: $c := m^e \mod n$
- Decryption $m' := c^d \mod n$

RSA-Assumption

$\approx$ *It is impractical for a given public key to extract a randomly choosen plaintext from a ciphertext.*

How not to do it (RSA)

RSA

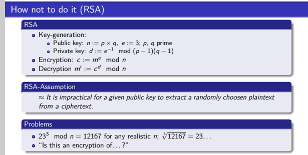
- Key-generation:
  - Public key: $n := p \times q,\ e := 3;\ p,\ q$ prime
  - Private key: $d := e^{-1} \mod (p-1)(q-1)$
- Encryption: $c := m^e \mod n$
- Decryption $m' := c^d \mod n$

RSA-Assumption

$\approx$ *It is impractical for a given public key to extract a randomly choosen plaintext from a ciphertext.*

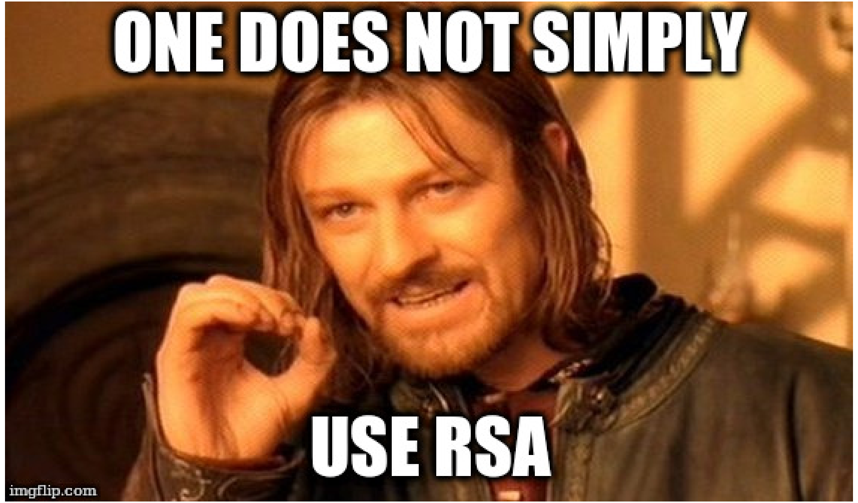
Problems

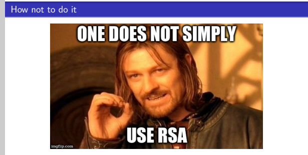
- $23^3 \mod n = 12167$ for any realistic n; $\sqrt[3]{12167} = 23\ldots$
- “Is this an encryption of…?”



How not to do it (RSA)
RSA
- Key-generation:
  - Public key: $n := p \times q,\ e := 3;\ p,\ q$ prime
  - Private key: $d := e^{-1} \mod (p-1)(q-1)$
- Encryption: $c := m^e \mod n$
- Decryption $m' := c^d \mod n$
RSA-Assumption
$\approx$ *It is impractical for a given public key to extract a randomly choosen plaintext from a ciphertext.*
Problems
- $23^3 \mod n = 12167$ for any realistic n; $\sqrt[3]{12167} = 23\ldots$
- “Is this an encryption of…?”

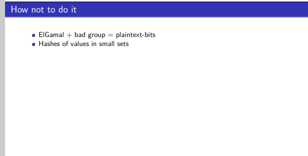
How not to do it




How not to do it

How not to do it

- ElGamal + bad group = plaintext-bits
- Hashes of values in small sets


How not to do it

- ElGamal + bad group = plaintext-bits
- Hashes of values in small sets

How not to do it

- ElGamal + bad group = plaintext-bits
- Hashes of values in small sets

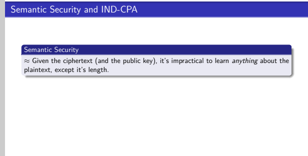
How not to do it

- ElGamal + bad group = plaintext-bits
- Hashes of values in small sets

Semantic Security and IND-CPA

Semantic Security

$\approx$ Given the ciphertext (and the public key), it's impractical to learn *anything* about the plaintext, except it's length.



Semantic Security and IND-CPA

Semantic Security

$\approx$ Given the ciphertext (and the public key), it's impractical to learn *anything* about the plaintext, except it's length.

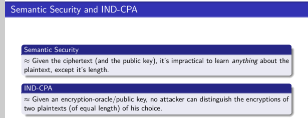
Semantic Security and IND-CPA

Semantic Security

≈ Given the ciphertext (and the public key), it's impractical to learn *anything* about the plaintext, except it's length.

IND-CPA

≈ Given an encryption-oracle/public key, no attacker can distinguish the encryptions of two plaintexts (of equal length) of his choice.



Semantic Security and IND-CPA

Semantic Security
≈ Given the ciphertext (and the public key), it's impractical to learn *anything* about the plaintext, except it's length.

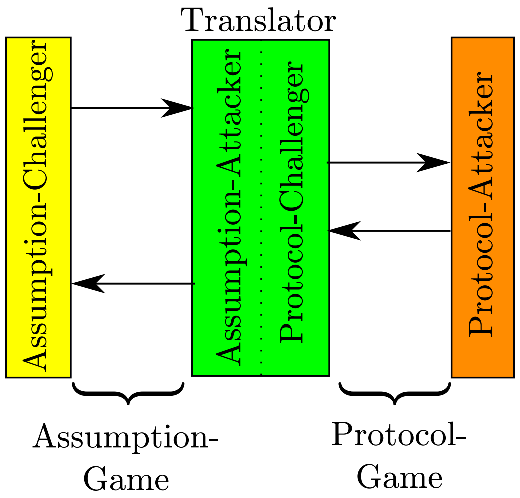
IND-CPA
≈ Given an encryption-oracle/public key, no attacker can distinguish the encryptions of two plaintexts (of equal length) of his choice.

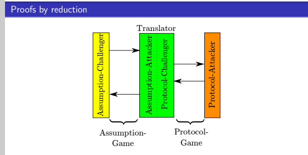
Out-of-model-attacks

Out-of-model-attacks

- We already mentioned Bleichenbacher
- There will be more. . .
- Side-channel-attacks
- Composition might give evil environments
- Often the hardest part in all of cryptography

Proofs by reduction



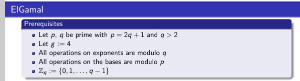

Proofs by reduction

- if we can give a translator, these assumptions contradict
- so either the assumption is wrong, or there is no adversary.

ElGamal

Prerequisites

- Let p, q be prime with $p = 2q + 1$ and $q > 2$
- Let $g := 4$
- All operations on exponents are modulo q
- All operations on the bases are modulo p
- $\mathbb{Z}_q := \{0, 1, \ldots, q - 1\}$



ElGamal

Prerequisites

- Let p, q be prime with $p = 2q + 1$ and $q > 2$
- Let $g := 4$
- All operations on exponents are modulo q
- All operations on the bases are modulo p
- $\mathbb{Z}_q := \{0, 1, \ldots, q - 1\}$

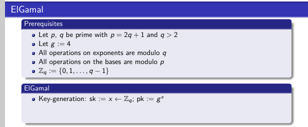
ElGamal

Prerequisites

- Let p, q be prime with $p = 2q + 1$ and $q > 2$
- Let $g := 4$
- All operations on exponents are modulo q
- All operations on the bases are modulo p
- $\mathbb{Z}_q := \{0, 1, \dots, q - 1\}$

ElGamal

- Key-generation: $\mathrm{sk} := x \leftarrow \mathbb{Z}_q$; $\mathrm{pk} := g^x$



ElGamal

Prerequisites

- Let p, q be prime with $p = 2q + 1$ and $q > 2$
- Let $g := 4$
- All operations on exponents are modulo q
- All operations on the bases are modulo p
- $\mathbb{Z}_q := \{0, 1, \dots, q - 1\}$

ElGamal

- Key-generation: $\mathrm{sk} := x \leftarrow \mathbb{Z}_q$; $\mathrm{pk} := g^x$

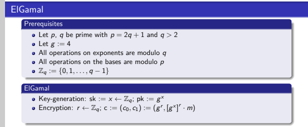
ElGamal

Prerequisites

- Let p, q be prime with $p = 2q + 1$ and $q > 2$
- Let $g := 4$
- All operations on exponents are modulo q
- All operations on the bases are modulo p
- $\mathbb{Z}_q := \{0, 1, \ldots, q - 1\}$

ElGamal

- Key-generation: sk $:= x \leftarrow \mathbb{Z}_q$; pk $:= g^x$
- Encryption: $r \leftarrow \mathbb{Z}_q$; c $:= (c_0, c_1) := (g^r, [g^x]^r \cdot m)$



ElGamal

Prerequisites

- Let p, q be prime with $p = 2q + 1$ and $q > 2$
- Let $g := 4$
- All operations on exponents are modulo q
- All operations on the bases are modulo p
- $\mathbb{Z}_q := \{0, 1, \ldots, q - 1\}$

ElGamal

- Key-generation: sk $:= x \leftarrow \mathbb{Z}_q$; pk $:= g^x$
- Encryption: $r \leftarrow \mathbb{Z}_q$; c $:= (c_0, c_1) := (g^r, [g^x]^r \cdot m)$

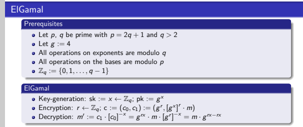
ElGamal

Prerequisites

- Let p, q be prime with $p = 2q + 1$ and $q > 2$
- Let $g := 4$
- All operations on exponents are modulo q
- All operations on the bases are modulo p
- $\mathbb{Z}_q := \{0, 1, \ldots, q - 1\}$

ElGamal

- Key-generation: $\mathrm{sk} := x \leftarrow \mathbb{Z}_q$; $\mathrm{pk} := g^x$
- Encryption: $r \leftarrow \mathbb{Z}_q$; $\mathrm{c} := (c_0, c_1) := (g^r, [g^x]^r \cdot m)$
- Decryption: $m' := c_1 \cdot [c_0]^{-x} = g^{rx} \cdot m \cdot [g^r]^{-x} = m \cdot g^{rx-rx}$



ElGamal

Prerequisites

- Let p, q be prime with $p = 2q + 1$ and $q > 2$
- Let $g := 4$
- All operations on exponents are modulo q
- All operations on the bases are modulo p
- $\mathbb{Z}_q := \{0, 1, \ldots, q - 1\}$

ElGamal

- Key-generation: $\mathrm{sk} := x \leftarrow \mathbb{Z}_q$; $\mathrm{pk} := g^x$
- Encryption: $r \leftarrow \mathbb{Z}_q$; $\mathrm{c} := (c_0, c_1) := (g^r, [g^x]^r \cdot m)$
- Decryption: $m' := c_1 \cdot [c_0]^{-x} = g^{rx} \cdot m \cdot [g^r]^{-x} = m \cdot g^{rx-rx}$

ElGamal

Prerequisites

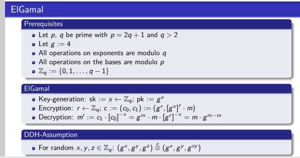
- Let p, q be prime with $p = 2q + 1$ and $q > 2$
- Let $g := 4$
- All operations on exponents are modulo q
- All operations on the bases are modulo p
- $\mathbb{Z}_q := \{0, 1, \ldots, q - 1\}$

ElGamal

- Key-generation: $\mathrm{sk} := x \leftarrow \mathbb{Z}_q$; $\mathrm{pk} := g^x$
- Encryption: $r \leftarrow \mathbb{Z}_q$; $\mathrm{c} := (c_0, c_1) := (g^r, [g^x]^r \cdot m)$
- Decryption: $m' := c_1 \cdot [c_0]^{-x} = g^{rx} \cdot m \cdot [g^r]^{-x} = m \cdot g^{rx-rx}$

DDH-Assumption

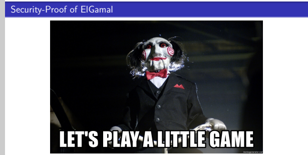
- For random $x, y, z \in \mathbb{Z}_q$: $(g^x, g^y, g^z) \overset{c}{\equiv} (g^x, g^y, g^{xy})$



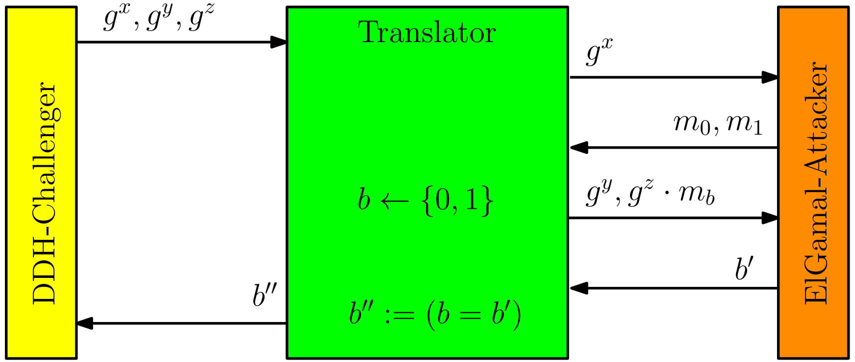
ElGamal
Prerequisites
- Let p, q be prime with $p = 2q + 1$ and $q > 2$
- Let $g := 4$
- All operations on exponents are modulo q
- All operations on the bases are modulo p
- $\mathbb{Z}_q := \{0, 1, \ldots, q - 1\}$
ElGamal
- Key-generation: $\mathrm{sk} := x \leftarrow \mathbb{Z}_q$; $\mathrm{pk} := g^x$
- Encryption: $r \leftarrow \mathbb{Z}_q$; $\mathrm{c} := (c_0, c_1) := (g^r, [g^x]^r \cdot m)$
- Decryption: $m' := c_1 \cdot [c_0]^{-x} = g^{rx} \cdot m \cdot [g^r]^{-x} = m \cdot g^{rx-rx}$
DDH-Assumption
- For random $x, y, z \in \mathbb{Z}_q$: $(g^x, g^y, g^z) \overset{c}{\equiv} (g^x, g^y, g^{xy})$

Security-Proof of ElGamal

Provable Security
└─Proofs and Modells
└─Security-Proof of ElGamal


Security-Proof of ElGamal

Security-Proof of ElGamal



DDH-Challenger

Translator

ElGamal-Attacker

g^x, g^y, g^z

g^x

m_0, m_1

$b \leftarrow \{0,1\}$

$g^y, g^z \cdot m_b$

b'

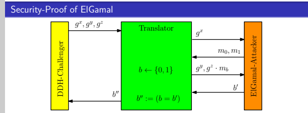
b''

$b'' := (b = b')$

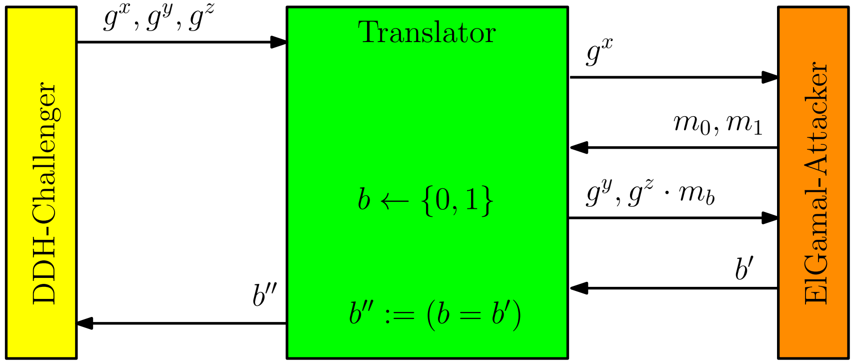
2018-12-29

Provable Security
└Proofs and Modells

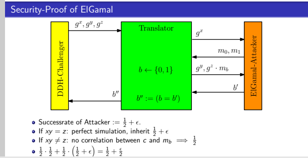
└Security-Proof of ElGamal



Security-Proof of ElGamal

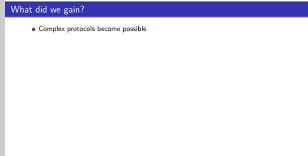


- Successrate of Attacker := $\frac{1}{2} + \epsilon$.
- If $xy = z$: perfect simulation, inherit $\frac{1}{2} + \epsilon$
- If $xy \neq z$: no correlation between c and m_b $\implies \frac{1}{2}$
- $\frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \left(\frac{1}{2} + \epsilon\right) = \frac{1}{2} + \frac{\epsilon}{2}$


Security-Proof of ElGamal
- Successrate of Attacker := $\frac{1}{2} + \epsilon$.
- If $xy = z$: perfect simulation, inherit $\frac{1}{2} + \epsilon$
- If $xy \neq z$: no correlation between c and m_b $\implies \frac{1}{2}$
- $\frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \left(\frac{1}{2} + \epsilon\right) = \frac{1}{2} + \frac{\epsilon}{2}$

What did we gain?

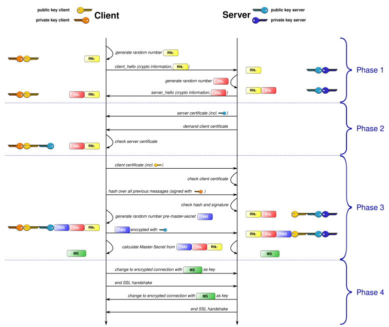
- Complex protocols become possible


What did we gain?

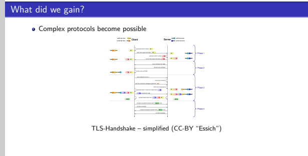
- Complex protocols become possible

What did we gain?

- Complex protocols become possible


public key client
private key client
Client
Server
public key server
private key server
generate random number RNc
RNc
client_hello (crypto information, RNc)
generate random number RNs
server_hello (crypto information, RNs)
RNs RNc
Phase 1
server certificate (incl.)
demand client certificate
check server certificate
Phase 2
client certificate (incl.)
check client certificate
hash over all previous messages (signed with)
check hash and signature
generate random number pre-master-secret PMS
PMS encrypted with
calculate Master-Secret from PMS RNs RNc
MS
Phase 3
change to encrypted connection with MS as key
end SSL handshake
change to encrypted connection with MS as key
end SSL handshake
Phase 4

TLS-Handshake – simplified (CC-BY “Essich”)

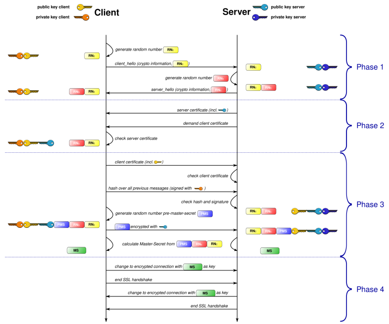

What did we gain?

- Complex protocols become possible

TLS-Handshake – simplified (CC-BY “Essich”)

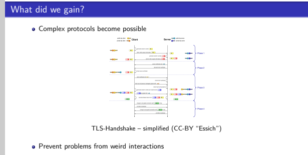
What did we gain?

- Complex protocols become possible


public key client
private key client
Client
Server
public key server
private key server
generate random number RNc
RNc
client_hello (crypto information, RNc)
RNc
generate random number RNs
server_hello (crypto information, RNs)
RNs RNc
RNc RNs
Phase 1
server certificate (incl.)
demand client certificate
check server certificate
RNs RNc
Phase 2
client certificate (incl.)
check client certificate
hash over all previous messages (signed with)
check hash and signature
generate random number pre-master-secret PMS
RNc RNs
PMS RNs RNc
PMS encrypted with
RNc RNs PMS
calculate Master-Secret from PMS RNs RNc
MS
MS
Phase 3
change to encrypted connection with MS as key
end SSL handshake
change to encrypted connection with MS as key
end SSL handshake
Phase 4

TLS-Handshake – simplified (CC-BY “Essich”)

- Prevent problems from weird interactions


What did we gain?

- Complex protocols become possible

TLS-Handshake – simplified (CC-BY “Essich”)

- Prevent problems from weird interactions

Security-models

Game-Based

- (Generally) easier proofs
- Often less intuitive meaning
- Does not scale

Security-models

Game-Based

- (Generally) easier proofs
- Often less intuitive meaning
- Does not scale

remove this

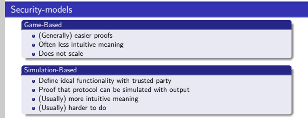
Security-models

Game-Based

- (Generally) easier proofs
- Often less intuitive meaning
- Does not scale

Simulation-Based

- Define ideal functionality with trusted party
- Proof that protocol can be simulated with output
- (Usually) more intuitive meaning
- (Usually) harder to do



Security-models

Game-Based
- (Generally) easier proofs
- Often less intuitive meaning
- Does not scale

Simulation-Based
- Define ideal functionality with trusted party
- Proof that protocol can be simulated with output
- (Usually) more intuitive meaning
- (Usually) harder to do

remove this

Security-models

Game-Based

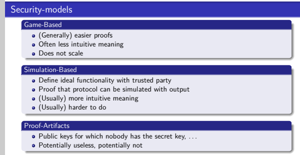
- (Generally) easier proofs
- Often less intuitive meaning
- Does not scale

Simulation-Based

- Define ideal functionality with trusted party
- Proof that protocol can be simulated with output
- (Usually) more intuitive meaning
- (Usually) harder to do

Proof-Artifacts

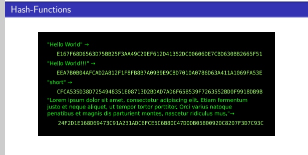
- Public keys for which nobody has the secret key, . . .
- Potentially useless, potentially not


Security-models
Game-Based
- (Generally) easier proofs
- Often less intuitive meaning
- Does not scale
Simulation-Based
- Define ideal functionality with trusted party
- Proof that protocol can be simulated with output
- (Usually) more intuitive meaning
- (Usually) harder to do
Proof-Artifacts
- Public keys for which nobody has the secret key, . . .
- Potentially useless, potentially not

remove this

Hash-Functions

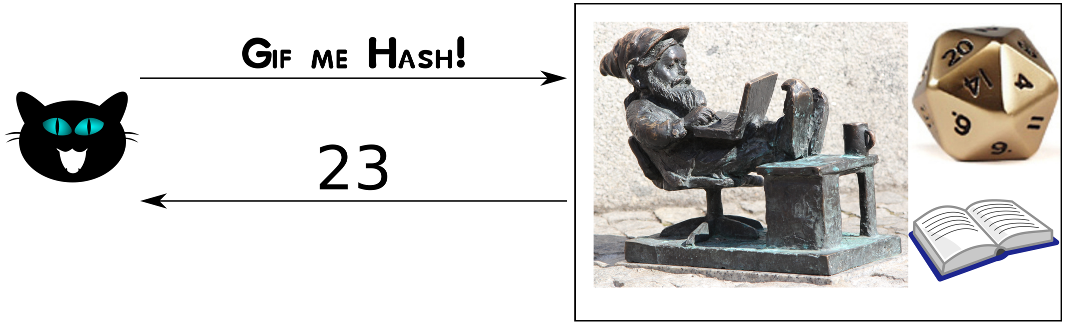
```
"Hello World" →
    E167F68D6563D75BB25F3AA49C29EF612D41352DC00606DE7CBD630BB2665F51
"Hello World!!!" →
    EEA7B0B04AFCAD2A812F1F8FB8B7A09B9E9C8D7010A0786D63A411A1069FA53E
"short" →
    CFCA535D38D7254948351E08713D2BDAD7AD6F65B539F7263552BD0F9918DB9B
"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam fermentum
justo et neque aliquet, ut tempor tortor porttitor. Orci varius natoque
penatibus et magnis dis parturient montes, nascetur ridiculus mus."→
    24F2D1E168D69473C91A231ADC6FCE5C6B80C47D0DB05800920C8207F3D7C93C
```



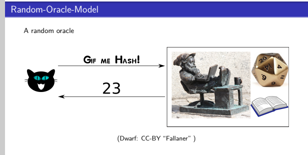
Used Hash-Function: SHA3-256

Random-Oracle-Model

A random oracle



(Dwarf: CC-BY "Fallaner")


Random-Oracle-Model
A random oracle
Gif me Hash!
23
(Dwarf: CC-BY "Fallaner")

- Hash functions are difficult to handle in proofs
- especially in an abstract way

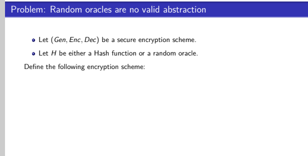
How a real Random oracle would look like:

- no one ever found a box with a dwarf
- those boxes would be difficult to handle
- ⇒ better use a hash function everyone can evaluate
- Problems:
  1. would be difficult to handle
  2. is not a valid abstraction

Problem: Random oracles are no valid abstraction

- Let (Gen, Enc, Dec) be a secure encryption scheme.
- Let H be either a Hash function or a random oracle.

Define the following encryption scheme:



Problem: Random oracles are no valid abstraction

- Let (Gen, Enc, Dec) be a secure encryption scheme.
- Let H be either a Hash function or a random oracle.

Define the following encryption scheme:

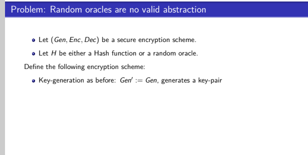
1. Note: Code execution attack is not the problem here.
2. assume that the attacker has a encryption oracle, i.e. he can force someone to
3. Lets construct a scheme which is secure in the ROM, but insecure for any Hash function
4. presented counterexample is dervied from one by Jonathan Katz

Problem: Random oracles are no valid abstraction

- Let (Gen, Enc, Dec) be a secure encryption scheme.
- Let H be either a Hash function or a random oracle.

Define the following encryption scheme:

- Key-generation as before: $Gen' := Gen$, generates a key-pair



Problem: Random oracles are no valid abstraction

- Let (Gen, Enc, Dec) be a secure encryption scheme.
- Let H be either a Hash function or a random oracle.

Define the following encryption scheme:

- Key-generation as before: $Gen' := Gen$, generates a key-pair

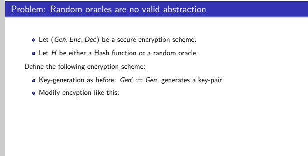
1. Note: Code execution attack is not the problem here.
2. assume that the attacker has a encryption oracle, i.e. he can force someone to
3. Lets construct a scheme which is secure in the ROM, but insecure for any Hash function
4. presented counterexample is dervied from one by Jonathan Katz

Problem: Random oracles are no valid abstraction

- Let (Gen, Enc, Dec) be a secure encryption scheme.
- Let H be either a Hash function or a random oracle.

Define the following encryption scheme:

- Key-generation as before: $Gen' := Gen$, generates a key-pair
- Modify encyption like this:



Problem: Random oracles are no valid abstraction

- Let (Gen, Enc, Dec) be a secure encryption scheme.
- Let H be either a Hash function or a random oracle.

Define the following encryption scheme:

- Key-generation as before: $Gen' := Gen$, generates a key-pair
- Modify encyption like this:

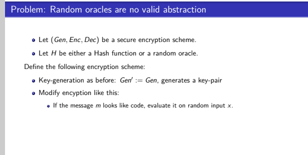
1. Note: Code execution attack is not the problem here.
2. assume that the attacker has a encryption oracle, i.e. he can force someone to
3. Lets construct a scheme which is secure in the ROM, but insecure for any Hash function
4. presented counterexample is dervied from one by Jonathan Katz

Problem: Random oracles are no valid abstraction

- Let (Gen, Enc, Dec) be a secure encryption scheme.
- Let H be either a Hash function or a random oracle.

Define the following encryption scheme:

- Key-generation as before: $Gen' := Gen$, generates a key-pair
- Modify encyption like this:
  - If the message m looks like code, evaluate it on random input x.



Problem: Random oracles are no valid abstraction

- Let (Gen, Enc, Dec) be a secure encryption scheme.
- Let H be either a Hash function or a random oracle.

Define the following encryption scheme:

- Key-generation as before: $Gen' := Gen$, generates a key-pair
- Modify encyption like this:
  - If the message m looks like code, evaluate it on random input x.

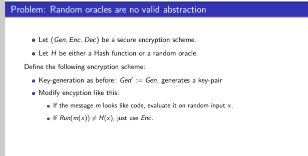
1. Note: Code execution attack is not the problem here.
2. assume that the attacker has a encryption oracle, i.e. he can force someone to
3. Lets construct a scheme which is secure in the ROM, but insecure for any Hash function
4. presented counterexample is dervied from one by Jonathan Katz

Problem: Random oracles are no valid abstraction

- Let (Gen, Enc, Dec) be a secure encryption scheme.
- Let H be either a Hash function or a random oracle.

Define the following encryption scheme:

- Key-generation as before: $Gen' := Gen$, generates a key-pair
- Modify encyption like this:
  - If the message m looks like code, evaluate it on random input x.
  - If $Run(m(x)) \neq H(x)$, just use Enc.



Problem: Random oracles are no valid abstraction

- Let (Gen, Enc, Dec) be a secure encryption scheme.
- Let H be either a Hash function or a random oracle.

Define the following encryption scheme:

- Key-generation as before: $Gen' := Gen$, generates a key-pair
- Modify encyption like this:
  - If the message m looks like code, evaluate it on random input x.
  - If $Run(m(x)) \neq H(x)$, just use Enc.

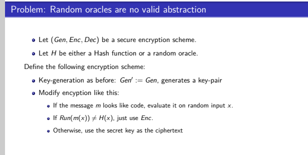
1. Note: Code execution attack is not the problem here.
2. assume that the attacker has a encryption oracle, i.e. he can force someone to
3. Lets construct a scheme which is secure in the ROM, but insecure for any Hash function
4. presented counterexample is dervied from one by Jonathan Katz

Problem: Random oracles are no valid abstraction

- Let (Gen, Enc, Dec) be a secure encryption scheme.
- Let H be either a Hash function or a random oracle.

Define the following encryption scheme:

- Key-generation as before: $Gen' := Gen$, generates a key-pair
- Modify encyption like this:
  - If the message m looks like code, evaluate it on random input x.
  - If $Run(m(x)) \neq H(x)$, just use Enc.
  - Otherwise, use the secret key as the ciphertext



Problem: Random oracles are no valid abstraction

- Let (Gen, Enc, Dec) be a secure encryption scheme.
- Let H be either a Hash function or a random oracle.

Define the following encryption scheme:

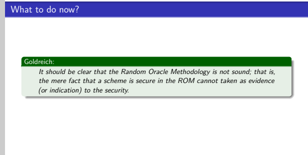
- Key-generation as before: $Gen' := Gen$, generates a key-pair
- Modify encyption like this:
  - If the message m looks like code, evaluate it on random input x.
  - If $Run(m(x)) \neq H(x)$, just use Enc.
  - Otherwise, use the secret key as the ciphertext

1. Note: Code execution attack is not the problem here.
2. assume that the attacker has a encryption oracle, i.e. he can force someone to
3. Lets construct a scheme which is secure in the ROM, but insecure for any Hash function
4. presented counterexample is dervied from one by Jonathan Katz

What to do now?

Goldreich:

> *It should be clear that the Random Oracle Methodology is not sound; that is, the mere fact that a scheme is secure in the ROM cannot taken as evidence (or indication) to the security.*


What to do now?

Goldreich:

It should be clear that the Random Oracle Methodology is not sound; that is, the mere fact that a scheme is secure in the ROM cannot taken as evidence (or indication) to the security.

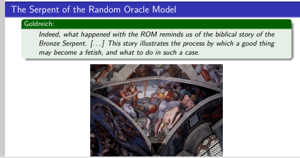
- authors came to harsh statements

  It should be clear that the Random Oracle Methodology is not sound; that is, the mere fact that a scheme is secure in the ROM cannot taken as evidence (or indication) to the security.

The Serpent of the Random Oracle Model

Goldreich:

> *Indeed, what happened with the ROM reminds us of the biblical story of the Bronze Serpent. [. . .] This story illustrates the process by which a good thing may become a fetish, and what to do in such a case.*



The Serpent of the Random Oracle Model
Goldreich:
Indeed, what happened with the ROM reminds us of the biblical story of the Bronze Serpent. [. . .] This story illustrates the process by which a good thing may become a fetish, and what to do in such a case.

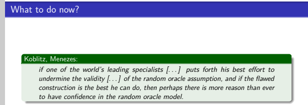
Indeed, what happened with the ROM reminds us of the biblical story of the Bronze Serpent. [. . .] This story illustrates the process by which a good thing may become a fetish, and what to do in such a case.

- spoiler alert: the snake had to be destroyed.
  - looking at the serpent healed snakebites; Hezekia destroyed it
- if you need to cite the bible as a cryptographer, you point may stand on feet of clay.

What to do now?

Koblitz, Menezes:

> *if one of the world's leading specialists [. . .] puts forth his best effort to undermine the validity [. . .] of the random oracle assumption, and if the flawed construction is the best he can do, then perhaps there is more reason than ever to have confidence in the random oracle model.*



What to do now?

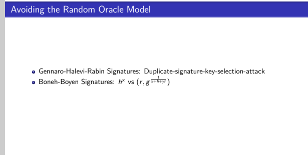
Koblitz, Menezes:

if one of the world's leading specialists [. . .] puts forth his best effort to undermine the validity [. . .] of the random oracle assumption, and if the flawed construction is the best he can do, then perhaps there is more reason than ever to have confidence in the random oracle model.

if one of the world's leading specialists [. . .] puts forth his best effort to undermine the validity [. . .] of the random oracle assumption, and if the flawed construction is the best he can do, then perhaps there is more reason than ever to have confidence in the random oracle model.

Avoiding the Random Oracle Model

- Gennaro-Halevi-Rabin Signatures: Duplicate-signature-key-selection-attack
- Boneh-Boyen Signatures: h^x vs $(r, g^{\frac{1}{x+h+yr}})$



Avoiding the Random Oracle Model

- Gennaro-Halevi-Rabin Signatures: Duplicate-signature-key-selection-attack
- Boneh-Boyen Signatures: h^x vs $(r, g^{\frac{1}{x+h+yr}})$

- For Gennaro-Halevi-Rabin Signatures it was shown that they have a strange Property: Duplicate-signature-key-selection-attacks.
- given a message and a signature, one can Calculate another key pair, such that the signature is valid for the same message under the new key
- Boneh-Boyen: Avoiding ROM made signatures twice as long and much more difficult to implement
- is this worth the effort?

Next:

- you might have noticed that we move more and more foreward into the beauty of proving-brain-fuck. For the next step I need to introduce another nice cryptographic tool called commitment schemes

A commitment scheme

Alice

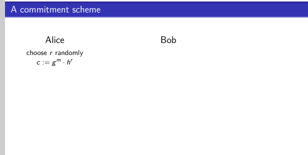
choose r randomly

$c := g^m \cdot h^r$

Bob

2018-12-29

Provable Security
└The struggles of Hash Functions
└A commitment scheme



A commitment scheme

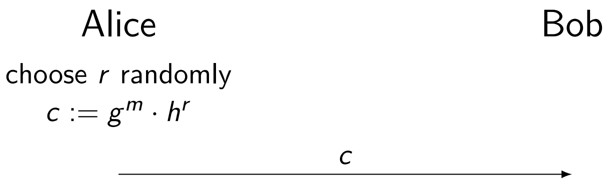
Alice

choose r randomly

$c := g^m \cdot h^r$

Bob

A commitment scheme



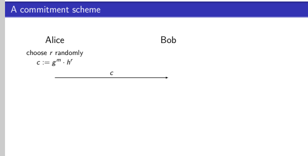
Alice

Bob

choose r randomly

$c := g^m \cdot h^r$

$\xrightarrow{c}$



A commitment scheme

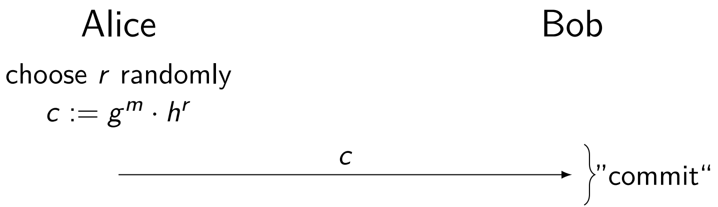
Alice Bob

choose r randomly

$c := g^m \cdot h^r$

$\xrightarrow{c}$

A commitment scheme



Alice

Bob

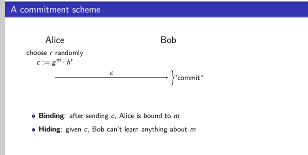
choose r randomly

$c := g^m \cdot h^r$

c

”commit“

- **Binding**: after sending c, Alice is bound to m
- **Hiding**: given c, Bob can't learn anything about m



A commitment scheme

Alice Bob

choose r randomly

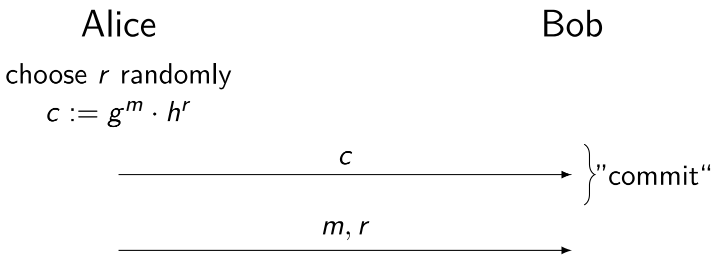
$c := g^m \cdot h^r$

c

”commit“

- **Binding**: after sending c, Alice is bound to m
- **Hiding**: given c, Bob can't learn anything about m

A commitment scheme



Alice Bob

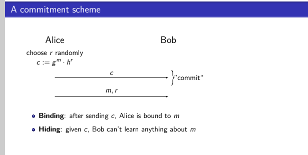
choose r randomly

$c := g^m \cdot h^r$

$$\xrightarrow{\quad c \quad} \Big\}\text{"commit"}$$

$$\xrightarrow{\quad m, r \quad}$$

- **Binding**: after sending c, Alice is bound to m
- **Hiding**: given c, Bob can't learn anything about m



A commitment scheme

Alice Bob

choose r randomly

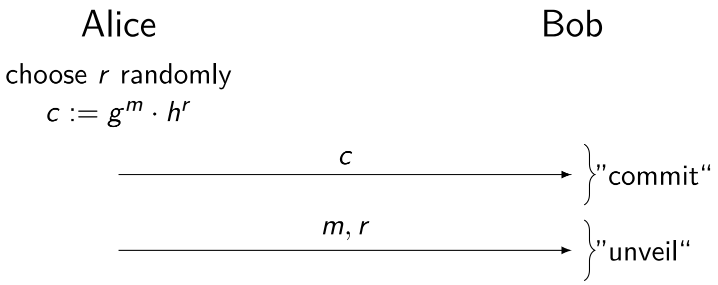
$c := g^m \cdot h^r$

c "commit"

m, r

- **Binding**: after sending c, Alice is bound to m
- **Hiding**: given c, Bob can't learn anything about m

A commitment scheme



Alice Bob

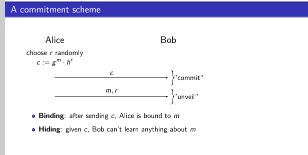
choose r randomly

$c := g^m \cdot h^r$

$\xrightarrow{c}$ ”commit“

$\xrightarrow{m, r}$ ”unveil“

- **Binding**: after sending c, Alice is bound to m
- **Hiding**: given c, Bob can't learn anything about m



A commitment scheme

Alice Bob

choose r randomly

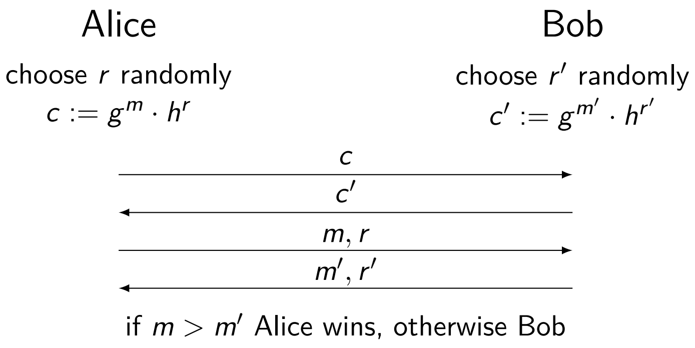
$c := g^m \cdot h^r$

$\xrightarrow{c}$ ”commit“

$\xrightarrow{m, r}$ ”unveil“

- **Binding**: after sending c, Alice is bound to m
- **Hiding**: given c, Bob can't learn anything about m

It's secure, but . . .



Alice

choose r randomly

$c := g^m \cdot h^r$

Bob

choose r' randomly

$c' := g^{m'} \cdot h^{r'}$

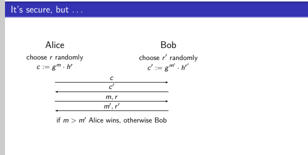
$\xrightarrow{c}$

$\xleftarrow{c'}$

$\xrightarrow{m, r}$

$\xleftarrow{m', r'}$

if $m > m'$ Alice wins, otherwise Bob



- auction szenario
- both parties commit on a price they want to pay
- higher value wins

Next:

- If Alice interacts with an evil party Mallory...

It's secure, but ...

Alice Mallory

choose r randomly

$c := g^m \cdot h^r$

$$\xrightarrow{\quad c \quad}$$

$$\xleftarrow{\quad c' := g \cdot c \quad}$$

$$\xrightarrow{\quad m, r \quad}$$

$$\xleftarrow{\quad m+1, r \quad}$$

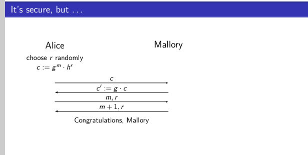
Congratulations, Mallory

2018-12-29

Provable Security

└The struggles of Hash Functions

└It's secure, but ...



It's secure, but ...

Alice Mallory

choose r randomly

$c := g^m \cdot h^r$

c

$c' := g \cdot c$

m, r

$m+1, r$

Congratulations, Mallory

Composability

- Security definitions should contain *all* imaginable properties
- A protocol should be *secure*, regardless of the context.

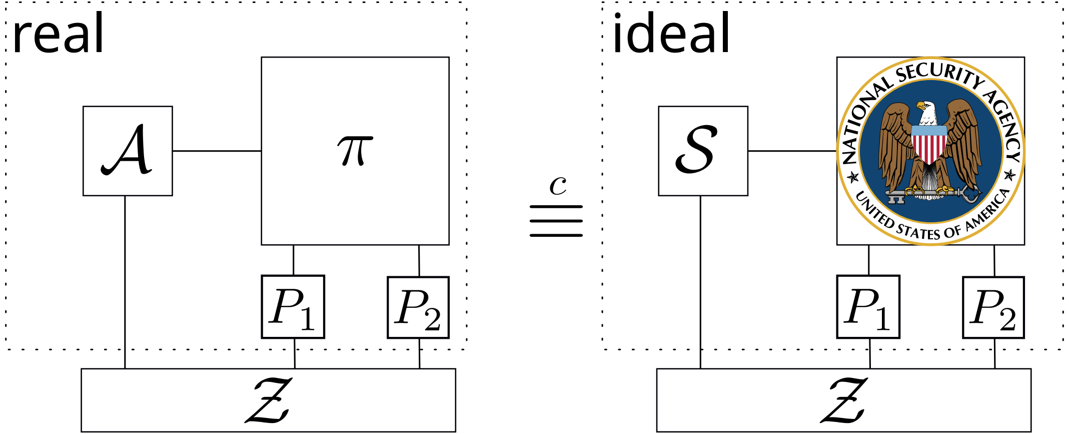
2018-12-29

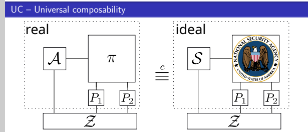
Composability

- Security definitions should contain *all* imaginable properties
- A protocol should be *secure*, regardless of the context.

There is a proving framework that offers this!

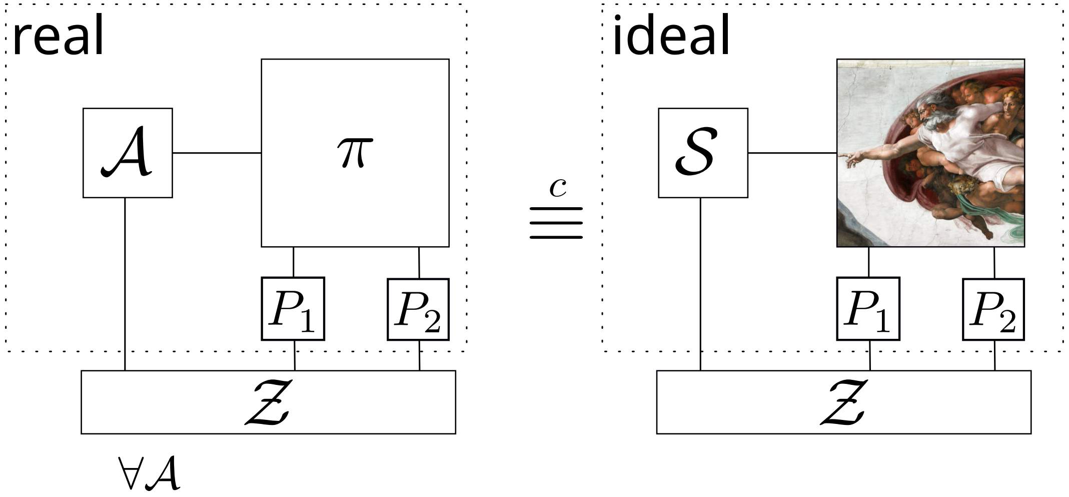
UC – Universal composability

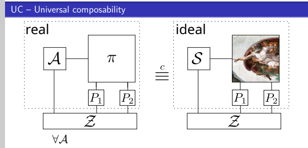




- Explain: F, Z, A, S

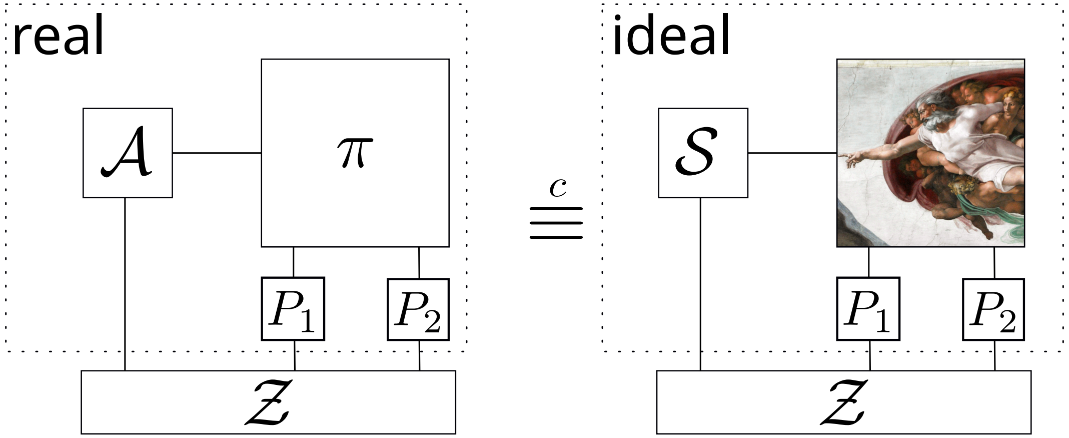
UC – Universal composability



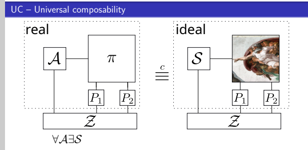

UC – Universal composability

- Explain: F, Z, A, S

UC – Universal composability

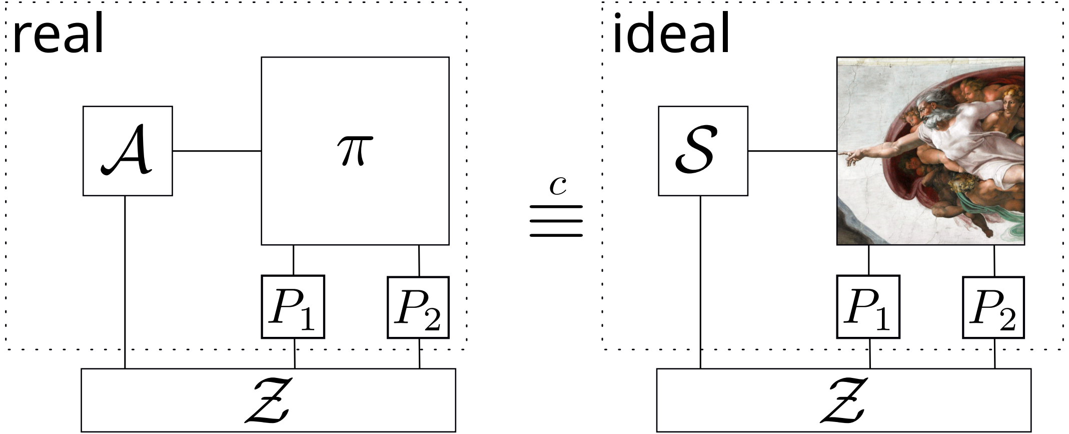


$\forall \mathcal{A} \exists \mathcal{S}$

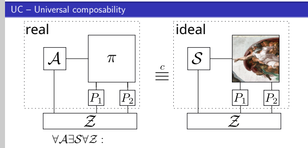


- Explain: F, Z, A, S

UC – Universal composability

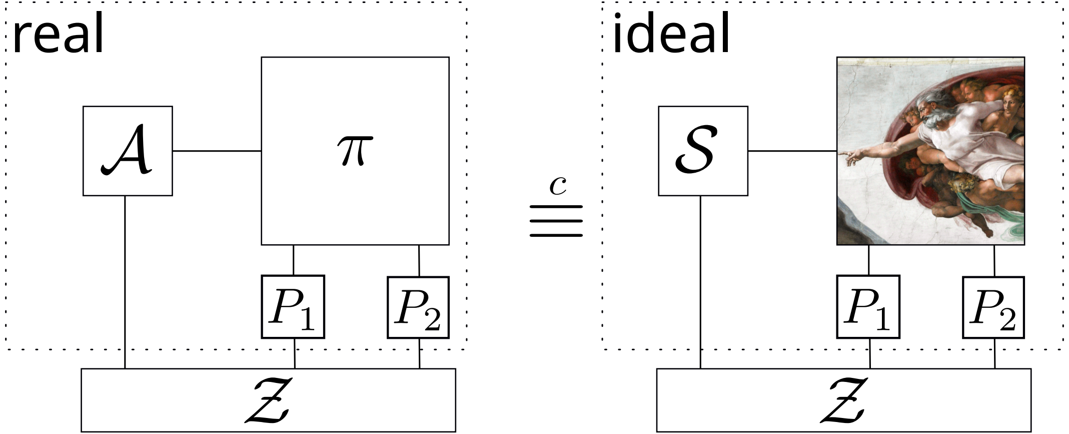


$\forall \mathcal{A} \exists \mathcal{S} \forall \mathcal{Z}$:

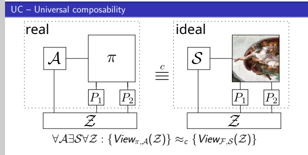


- Explain: F, Z, A, S

UC – Universal composability

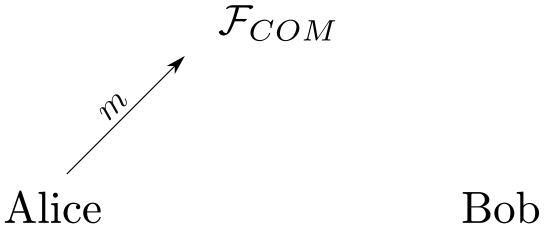


$$\forall \mathcal{A} \exists \mathcal{S} \forall \mathcal{Z} : \{View_{\pi,\mathcal{A}}(\mathcal{Z})\} \approx_c \{View_{\mathcal{F},\mathcal{S}}(\mathcal{Z})\}$$



- Explain: F, Z, A, S

$\mathcal{F}_{Com}$



$\mathcal{F}_{COM}$

m

Alice

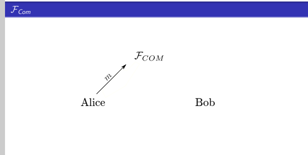
Bob

2018-12-29

Provable Security

└Universal composability

└$\mathcal{F}_{Com}$



$\mathcal{F}_{Com}$

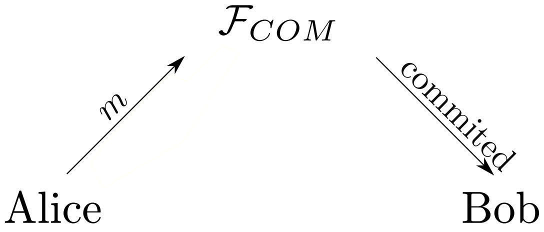
$\mathcal{F}_{COM}$

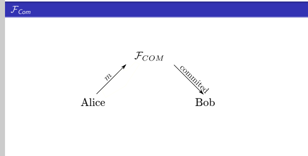
m

Alice

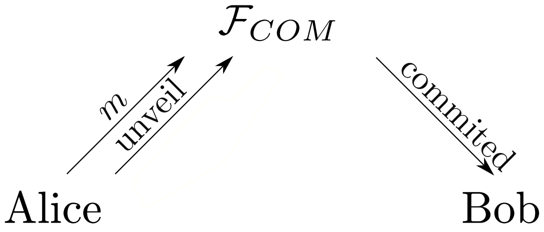
Bob

$\mathcal{F}_{Com}$



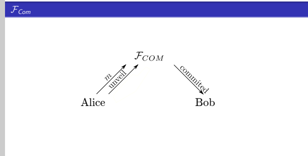

$\mathcal{F}_{Com}$
$\mathcal{F}_{COM}$
m
commited
Alice
Bob

$\mathcal{F}_{Com}$

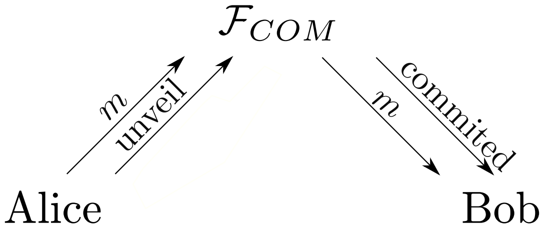


2018-12-29

└$\mathcal{F}_{Com}$


$\mathcal{F}_{Com}$

$\mathcal{F}_{Com}$



$\mathcal{F}_{COM}$

m

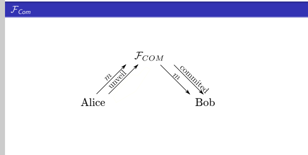
unveil

m

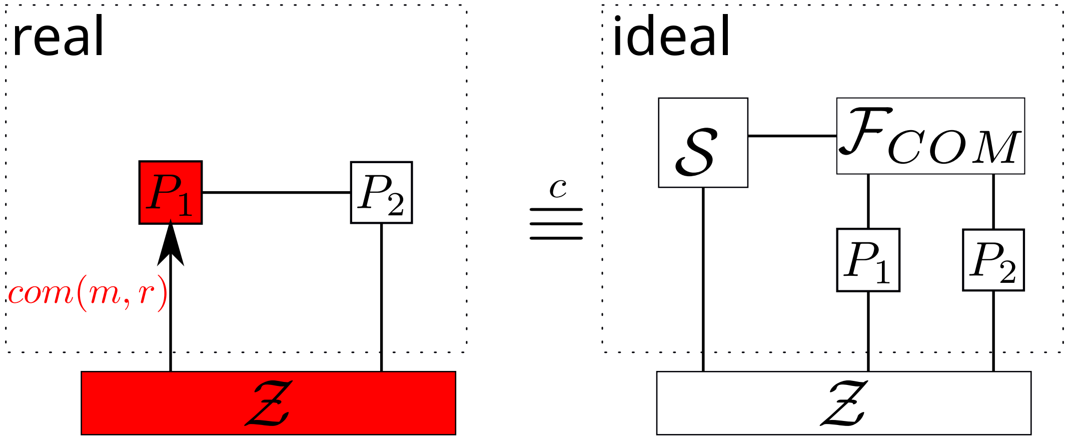
commited

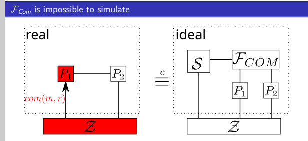
Alice

Bob



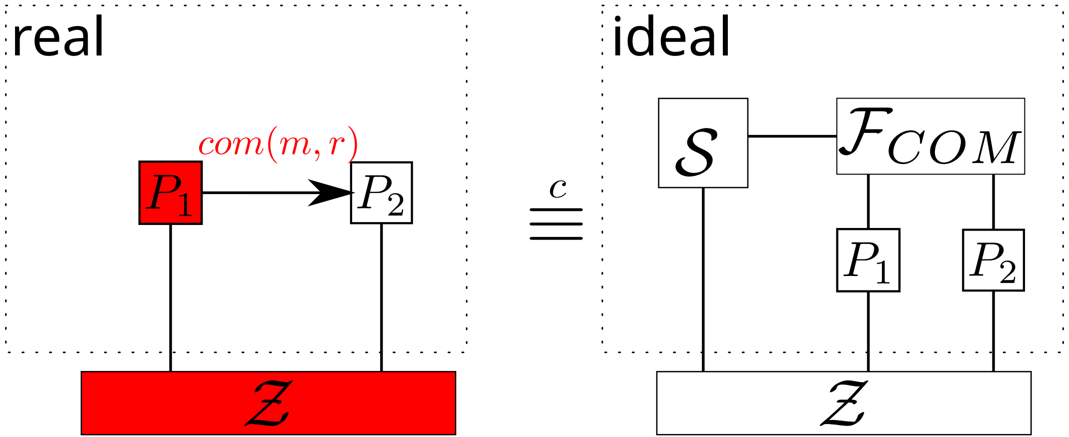
$\mathcal{F}_{Com}$ is impossible to simulate



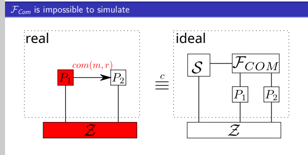


- attacker and environment are working together
- simulator wants to mimic the atttacker such that Z can't distinguish

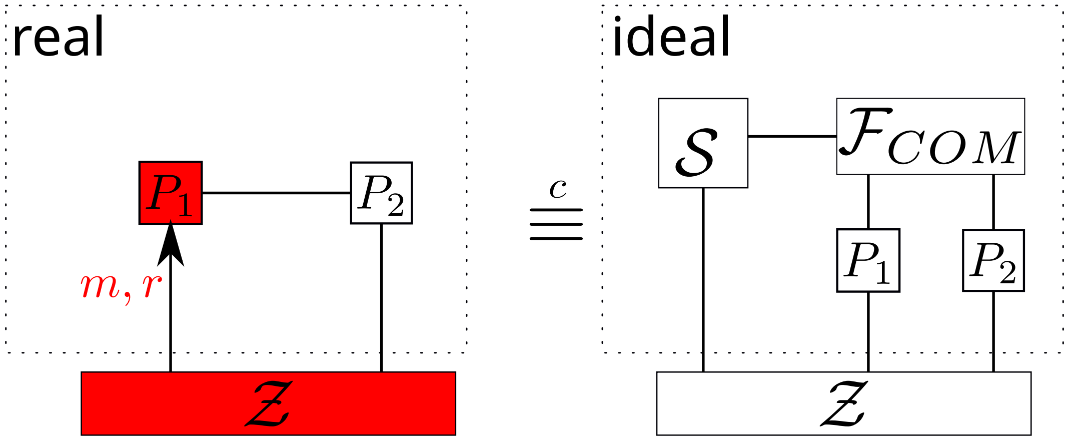
$\mathcal{F}_{Com}$ is impossible to simulate



└$\mathcal{F}_{Com}$ is impossible to simulate



$\mathcal{F}_{Com}$ is impossible to simulate

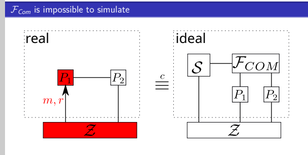


2018-12-29

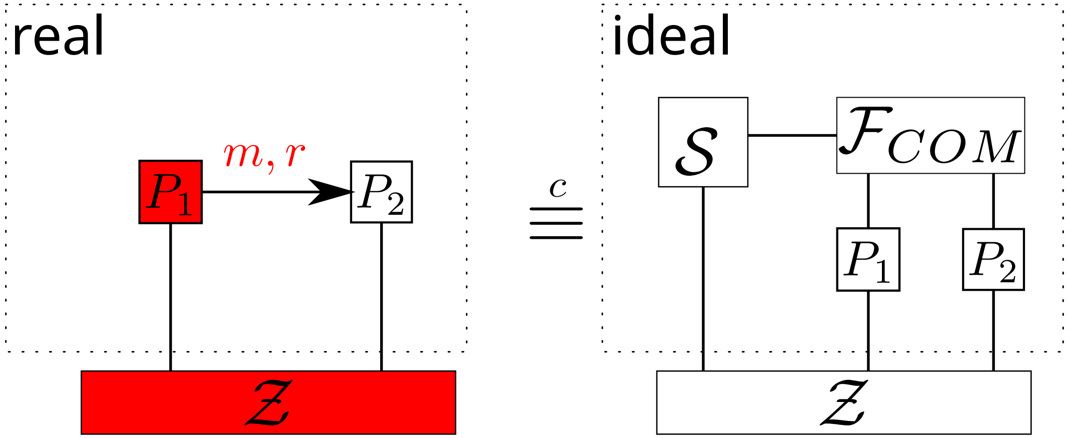
Provable Security

└Universal composability

└$\mathcal{F}_{Com}$ is impossible to simulate

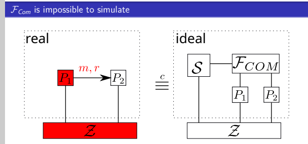

$\mathcal{F}_{Com}$ is impossible to simulate

$\mathcal{F}_{Com}$ is impossible to simulate

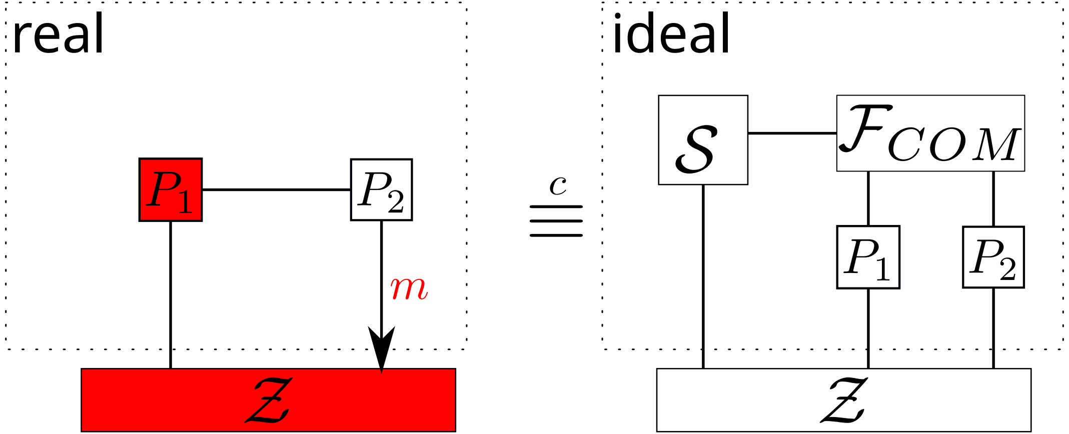


2018-12-29

└$\mathcal{F}_{Com}$ is impossible to simulate

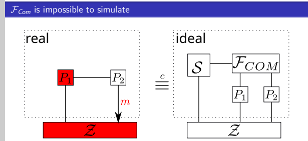


$\mathcal{F}_{Com}$ is impossible to simulate

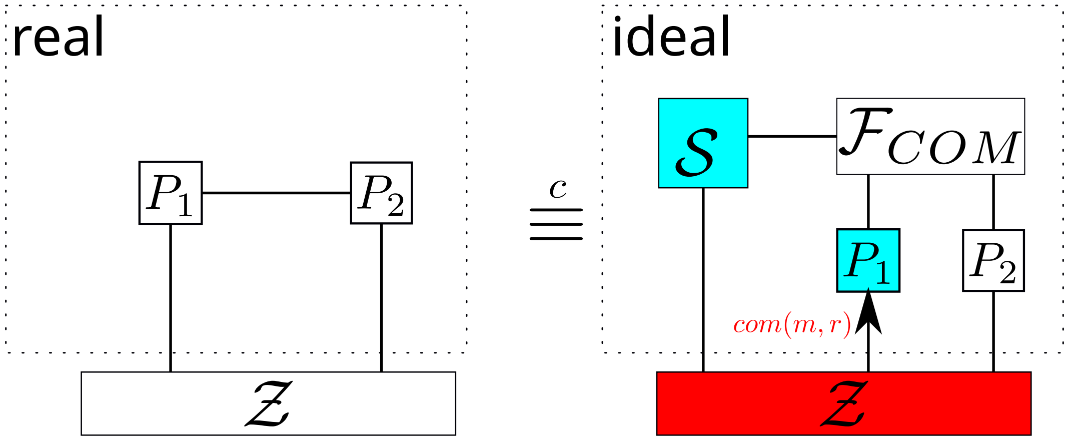


2018-12-29

Provable Security
└Universal composability
└$\mathcal{F}_{Com}$ is impossible to simulate

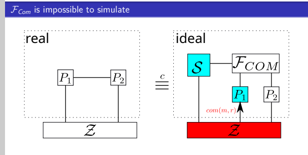

$\mathcal{F}_{Com}$ is impossible to simulate

$\mathcal{F}_{Com}$ is impossible to simulate

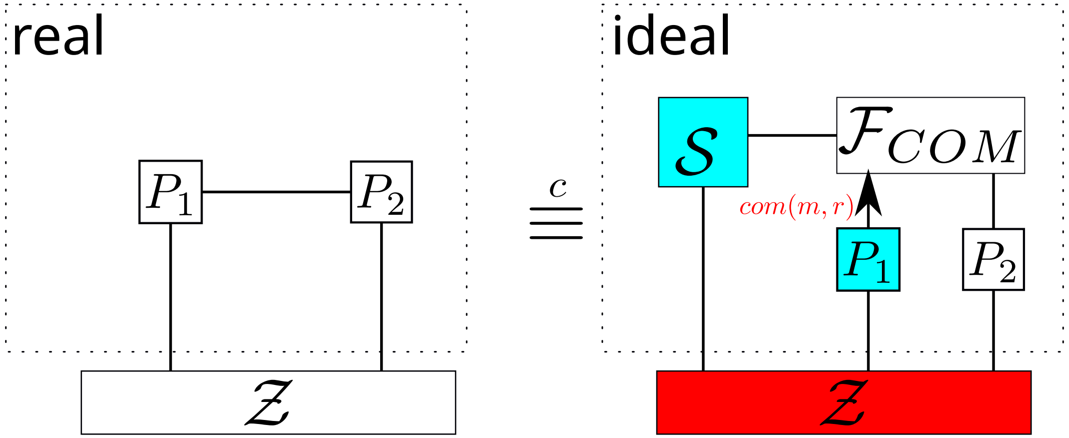


2018-12-29

Provable Security
└Universal composability
└$\mathcal{F}_{Com}$ is impossible to simulate

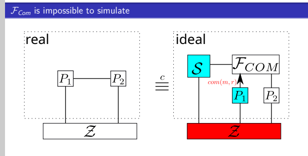


$\mathcal{F}_{Com}$ is impossible to simulate

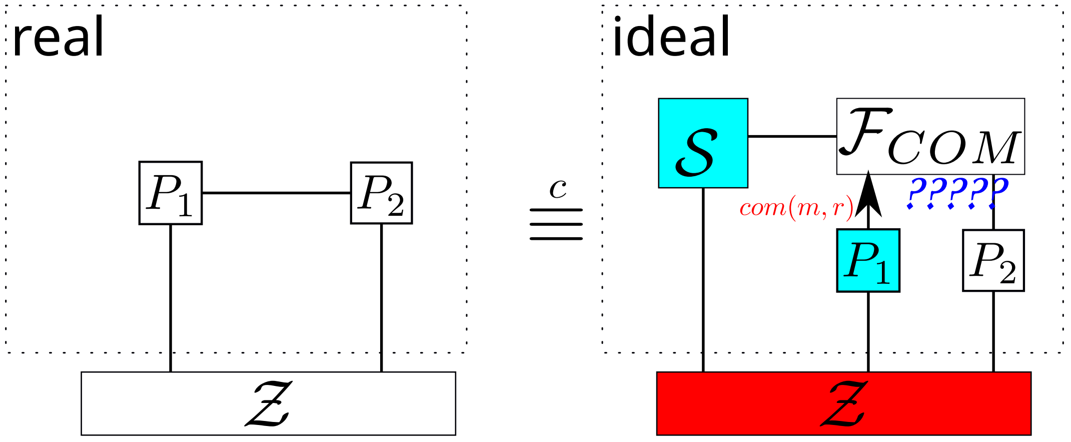


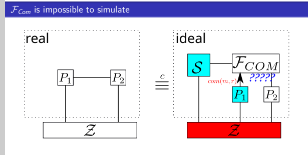
2018-12-29

Provable Security
└Universal composability
└$\mathcal{F}_{Com}$ is impossible to simulate



$\mathcal{F}_{Com}$ is impossible to simulate

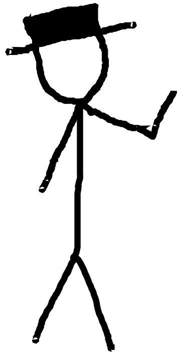




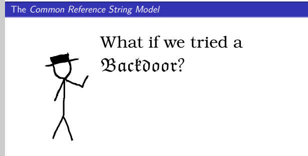
- Problem: $\mathcal{S}$ must provide a transcript
-
- A similar proof exists for the binding property.
-

⇒ No protocol can ever realize $\mathcal{F}_{Com}$.

The *Common Reference String Model*



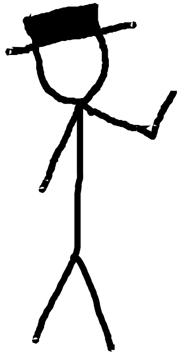
What if we tried a Backdoor?



The *Common Reference String Model*

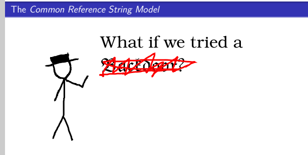
What if we tried a Backdoor?

The *Common Reference String Model*



What if we tried a

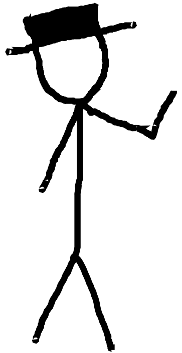
~~Backdoor~~?



The *Common Reference String Model*

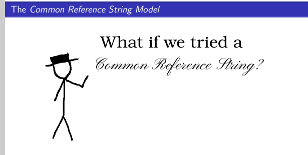
What if we tried a
~~Backdoor~~?

The *Common Reference String Model*



What if we tried a

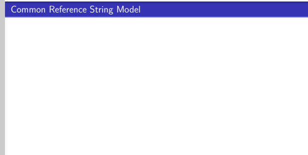
Common Reference String?



The *Common Reference String Model*

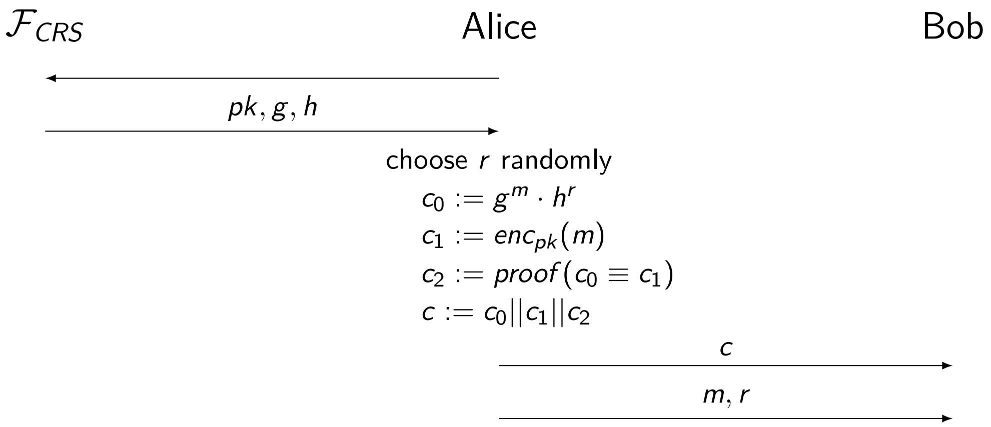
What if we tried a
Common Reference String?

Common Reference String Model



- Real: have a CRS
- Ideal: $\mathcal{S}$ simulates CRS

A *secure* Commitment Scheme



$\mathcal{F}_{CRS}$ Alice Bob

pk, g, h

choose r randomly

$c_0 := g^m \cdot h^r$

$c_1 := enc_{pk}(m)$

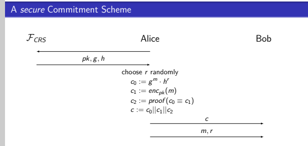
$c_2 := proof(c_0 \equiv c_1)$

$c := c_0||c_1||c_2$

c

m, r

Provable Security
└Universal composability
└A *secure* Commitment Scheme



A *secure* Commitment Scheme

$\mathcal{F}_{CRS}$ Alice Bob

pk, g, h

choose r randomly

$c_0 := g^m \cdot h^r$

$c_1 := enc_{pk}(m)$

$c_2 := proof(c_0 \equiv c_1)$

$c := c_0||c_1||c_2$

c

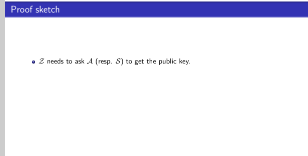
m, r

Proof sketch

- $\mathcal{Z}$ needs to ask $\mathcal{A}$ (resp. $\mathcal{S}$) to get the public key.

2018-12-29

Provable Security
└Universal composability
└Proof sketch

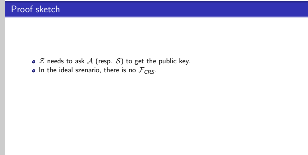


Proof sketch

- $\mathcal{Z}$ needs to ask $\mathcal{A}$ (resp. $\mathcal{S}$) to get the public key.

Proof sketch

- $\mathcal{Z}$ needs to ask $\mathcal{A}$ (resp. $\mathcal{S}$) to get the public key.
- In the ideal szenario, there is no $\mathcal{F}_{CRS}$.

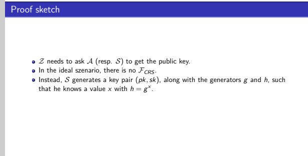


Proof sketch

- $\mathcal{Z}$ needs to ask $\mathcal{A}$ (resp. $\mathcal{S}$) to get the public key.
- In the ideal szenario, there is no $\mathcal{F}_{CRS}$.

Proof sketch

- $\mathcal{Z}$ needs to ask $\mathcal{A}$ (resp. $\mathcal{S}$) to get the public key.
- In the ideal szenario, there is no $\mathcal{F}_{CRS}$.
- Instead, $\mathcal{S}$ generates a key pair (pk, sk), along with the generators g and h, such that he knows a value x with $h = g^x$.

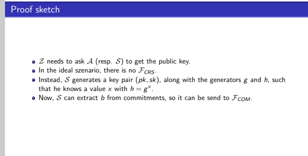


Proof sketch

- $\mathcal{Z}$ needs to ask $\mathcal{A}$ (resp. $\mathcal{S}$) to get the public key.
- In the ideal szenario, there is no $\mathcal{F}_{CRS}$.
- Instead, $\mathcal{S}$ generates a key pair (pk, sk), along with the generators g and h, such that he knows a value x with $h = g^x$.

Proof sketch

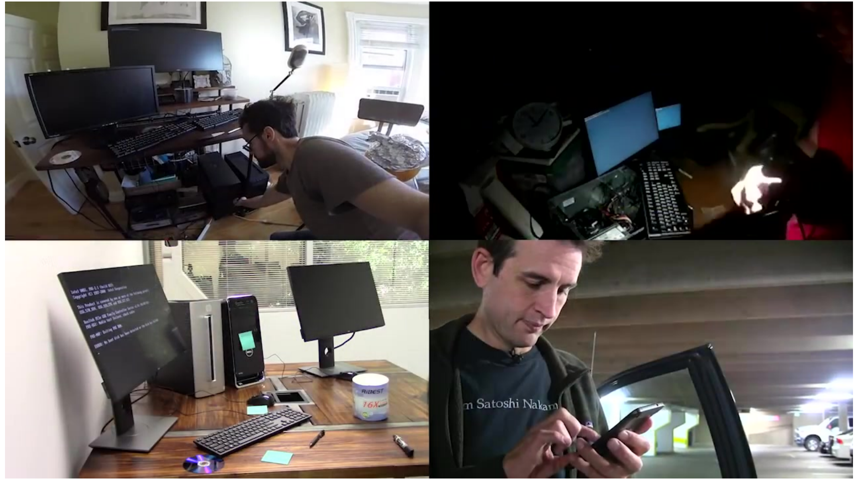
- $\mathcal{Z}$ needs to ask $\mathcal{A}$ (resp. $\mathcal{S}$) to get the public key.
- In the ideal szenario, there is no $\mathcal{F}_{CRS}$.
- Instead, $\mathcal{S}$ generates a key pair (pk, sk), along with the generators g and h, such that he knows a value x with $h = g^x$.
- Now, $\mathcal{S}$ can extract b from commitments, so it can be send to $\mathcal{F}_{COM}$.



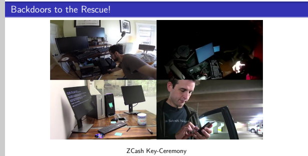
Proof sketch

- $\mathcal{Z}$ needs to ask $\mathcal{A}$ (resp. $\mathcal{S}$) to get the public key.
- In the ideal szenario, there is no $\mathcal{F}_{CRS}$.
- Instead, $\mathcal{S}$ generates a key pair (pk, sk), along with the generators g and h, such that he knows a value x with $h = g^x$.
- Now, $\mathcal{S}$ can extract b from commitments, so it can be send to $\mathcal{F}_{COM}$.

Backdoors to the Rescue!



ZCash Key-Ceremony


Backdoors to the Rescue!

ZCash Key-Ceremony

- ECDRBG (Elliptic Curve Deterministic Random Bit Generator) was only meant to provide extractability for UC-proofs, the NSA couldn't possibly have wanted to snoop

Too long, didn't watch

Rule #1

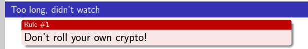
Don't roll your own crypto!

2018-12-29

Provable Security

└Universal composability

└Too long, didn't watch


Too long, didn't watch

Rule #1

Don't roll your own crypto!

Too long, didn't watch

Rule #1

Don't roll your own crypto!

what you should have learned

- Security is difficult, employ proofs
- Be aware of their limitations
- Good heuristics are better than nothing
- People might actually read simpler proofs

Too long, didn't watch

Rule #1

Don't roll your own crypto!

what you should have learned

- Security is difficult, employ proofs
- Be aware of their limitations
- Good heuristics are better than nothing
- People might actually read simpler proofs

Too long, didn't watch

Rule #1

Don't roll your own crypto!

what you should have learned

- Security is difficult, employ proofs
- Be aware of their limitations
- Good heuristics are better than nothing
- People might actually read simpler proofs

Questions?

Too long, didn't watch

Rule #1

Don't roll your own crypto!

what you should have learned

- Security is difficult, employ proofs
- Be aware of their limitations
- Good heuristics are better than nothing
- People might actually read simpler proofs

Questions?

Bonus-Slide: Security-Levels

Computational Security

- If Brute-Force is possible
- 128 Bit pre-quantum are fine

Statistical Security

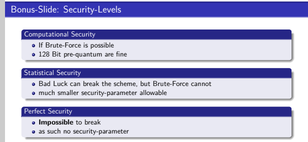
- Bad Luck can break the scheme, but Brute-Force cannot
- much smaller security-parameter allowable

Perfect Security

- **Impossible** to break
- as such no security-parameter

2018-12-29

Provable Security
└Universal composability
└Bonus-Slide: Security-Levels



Bonus-Slide: Security-Levels

Computational Security
- If Brute-Force is possible
- 128 Bit pre-quantum are fine

Statistical Security
- Bad Luck can break the scheme, but Brute-Force cannot
- much smaller security-parameter allowable

Perfect Security
- **Impossible** to break
- as such no security-parameter