Introduction to Provable Security

Models, Adversaries, Reductions

CRYPTOGRAPHY / CRYPTOLOGY

- “from Greek κρυπτός *kryptós*, "hidden, secret"; and γράφειν *graphein*, "writing", or -λογία *-logia*, "study", respectively”

- “is the practice and study of techniques for secure communication in the presence of third parties (called adversaries).”

Source : www.wikipedia.org

Some cryptographic goals

- Confidentiality
  - Content of conversation remains hidden
- Authenticity
  - Message is really sent by specific sender
- Integrity
  - Message has not been modified
- Privacy:
  - Sensitive (user) data remains hidden
- Covertcy
  - The fact that a conversation is taking place is hidden
- ….

Security by trial-and-error

- Identify goal (e.g. confidentiality in P2P networks)
- Design solution – the strategy:
  - Propose protocol
  - Search for an attack
  - If attack found, fix (go to first step)
  - After many iterations or some time, halt
- Output: resulting scheme
- Problems:
  - What is "many" iterations/ "some" time?
  - Some schemes take time to break: MD5, RC4…

Provable security

- Identify goal. Define security:
  - Syntax of the primitive: e.g. algorithms (KGen, Sign, Vf)
  - Adversary (e.g. can get signatures for arbitrary msgs.)
  - Security conditions (e.g. adv. can't sign fresh message)
- Propose a scheme (instantiate syntax)
- Define/choose security assumptions
  - Properties of primitives / number theoretical problems
- Prove security – 2 step algorithm:
  - Assume we can break security of scheme (adv. A)
  - Then build "Reduction" (adv. B) breaking assumption

Part II
The Provable Security Method

The essence of provable security

- Core question: what does "secure" mean?
  - "Secure encryption" vs. "Secure signature scheme"
- Say a scheme is secure against all known attacks
  - … will it be secure against a yet unknown attack?
- Step 1: Define your primitive (syntax)

Signature Scheme: algorithms (KGen, Sign, Vf)

* KGen(1^γ) outputs (sk, pk)
* Sign(sk,m) outputs S (prob.)
* Vf(pk,m,S) outputs 0 or 1 (det.)

THE ESSENCE OF PROVABLE SECURITY

- Step 2: Define your adversary

Adversaries A can: know public information: γ, pk
get no message/signature pair
get list of message/signature pairs
submit arbitrary message to sign

- Step 3: Define the security condition

Adversary A can output fresh (m,S) which verifies, with non-negligible probability (as a function of γ)

The essence of provable security

- Step 4: Propose a protocol

Instantiate the syntax given in Step 1.

E.g. give specific algorithms for KGen, Sign, Vf.

- Step 5: Choose security assumptions

For each primitive in the protocol, choose assumptions

- Security Assumptions (e.g. IND-CCA encryption)
- Number Theoretical Assumptions (e.g. DDH, RSA)

THE ESSENCE OF PROVABLE SECURITY

- Step 6: Prove security

For each property you defined in steps 1-3:

- Assume there exists an adversary A breaking that security property with some probability ε
- Construct reduction B breaking underlying assumption with probability $f(\varepsilon)$

How reductions work

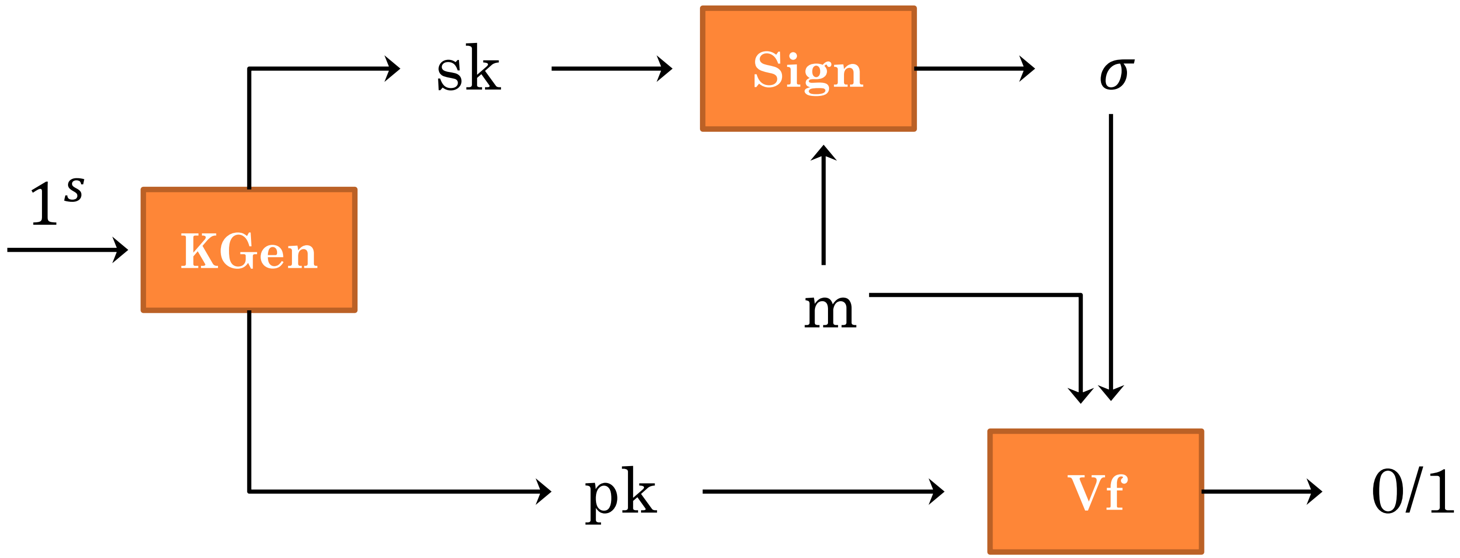
- Security assumptions are baseline
- Reasoning:
  - If our protocol/primitive is insecure, then the assumption is broken
  - But the assumption holds (by definition)
- Conclusion: The protocol cannot be insecure
- Caveat:
  - Say an assumption is broken (e.g. DDH easy to solve)
  - What does that say about our protocol?

We don’t know!

Part III
Assumptions

WE NEED COMPUTATIONAL ASSUMPTIONS

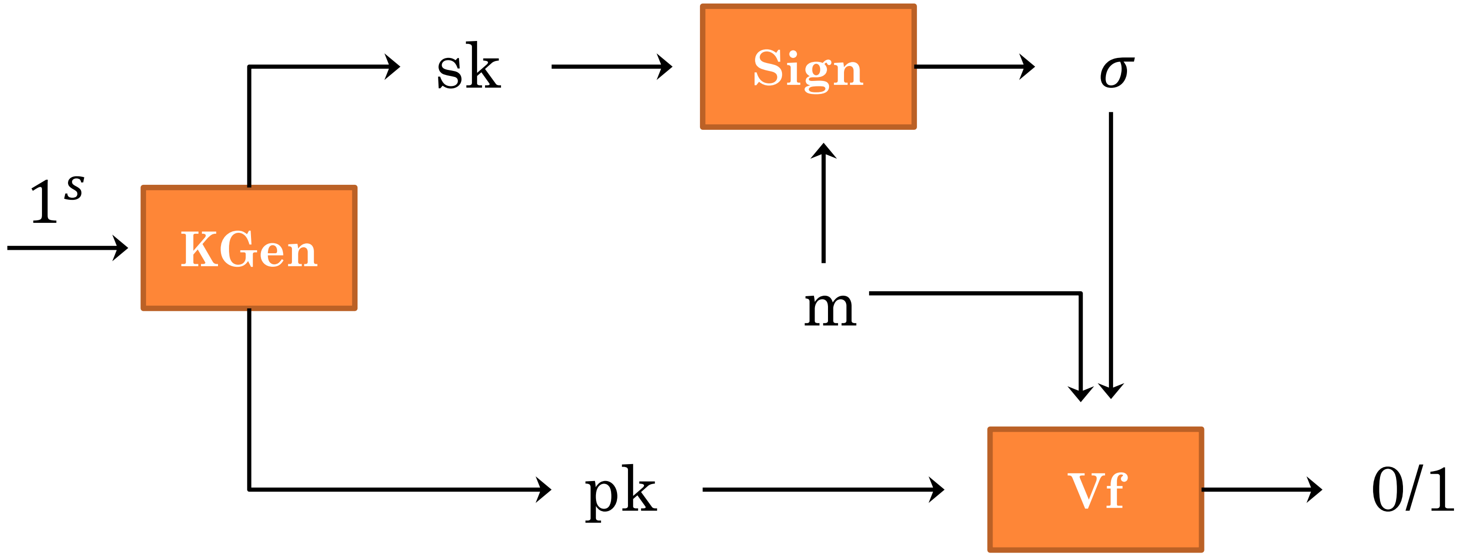
- Take our signature schemes (KGen, Sign, Vf)



- Correctness: if parameters are well generated, well-signed signatures always verify.

We need computational assumptions

- Take our signature schemes (KGen, Sign, Vf)

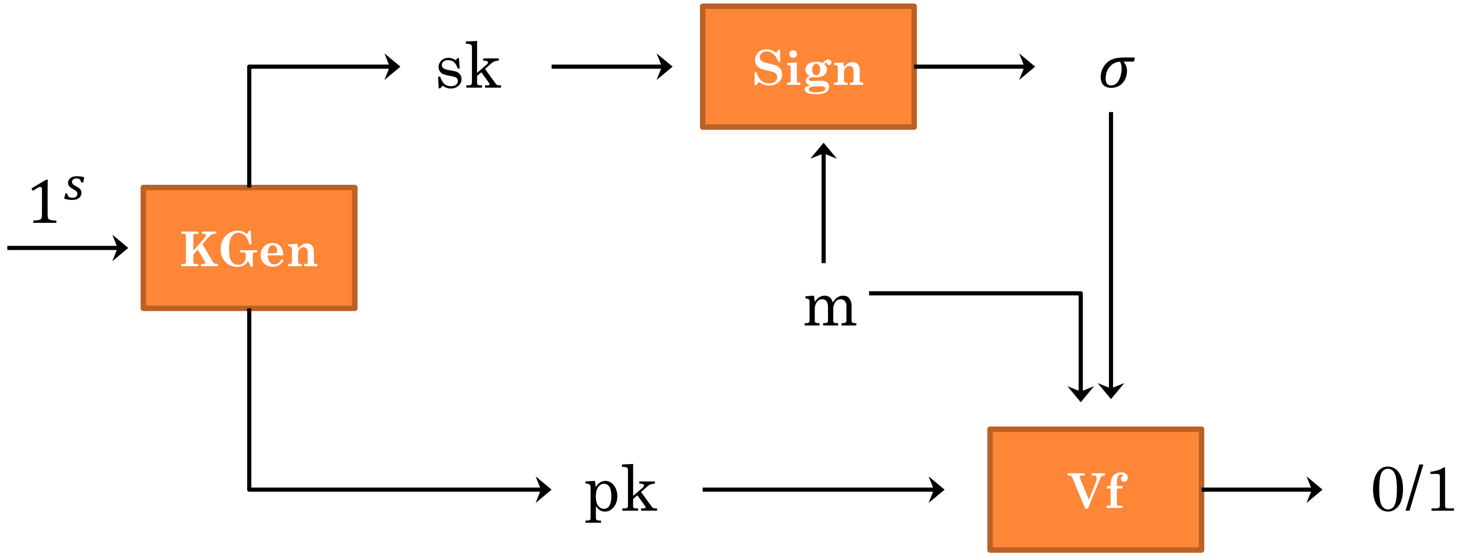


- Unforgeability: no adversary can produce signature for a fresh message m*

  But any A can guess sk with probability $1/2^{|sk|}$

WE NEED COMPUTATIONAL ASSUMPTIONS

- Take our signature schemes (KGen, Sign, Vf)



- Unforgeability: no adversary can produce signature for a fresh message m*

  And any A can guess valid σ with probability $^{1}/_{2^{|\sigma|}}$

Some Computational Assumptions

- Of the type: It is “hard” to compute x starting from y.
- How hard?
  - Usually no proof that the assumption holds
  - Mostly measured with respect to “best attack”
  - Sometimes average-case, sometimes worst-case
- Relation to other assumptions:
  - A 1 “→” A 2: break A 2 => break A 1 **stronger**
  - A 1 “←” A 2: break A 1 => break A 2 **weaker**
  - A 1 “⇔” A 2: both conditions hold **equivalent**

Examples: DLog, CDH, DDH

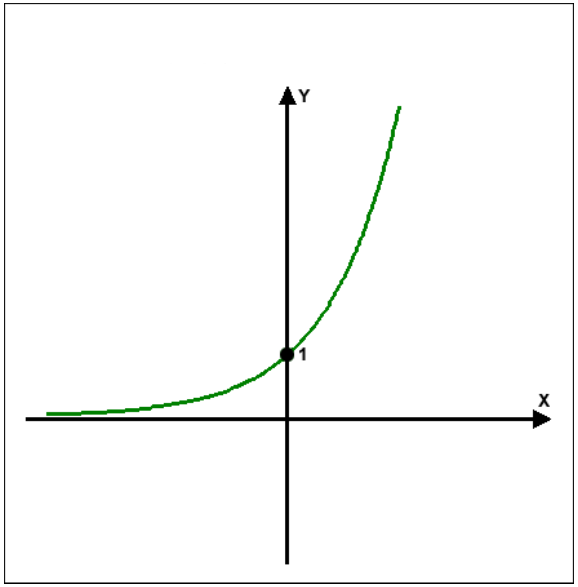
- Background:
  - Finite field $\mathbb{F}$, e.g. $\mathbb{Z}^*_p$ = {1, 2, … , p-1} for prime p
  - Multiplication, e.g. modulo p: $2(p-2) = 2p - 4 = p - 4$
  - Element g of prime order $q | (p-1)$:

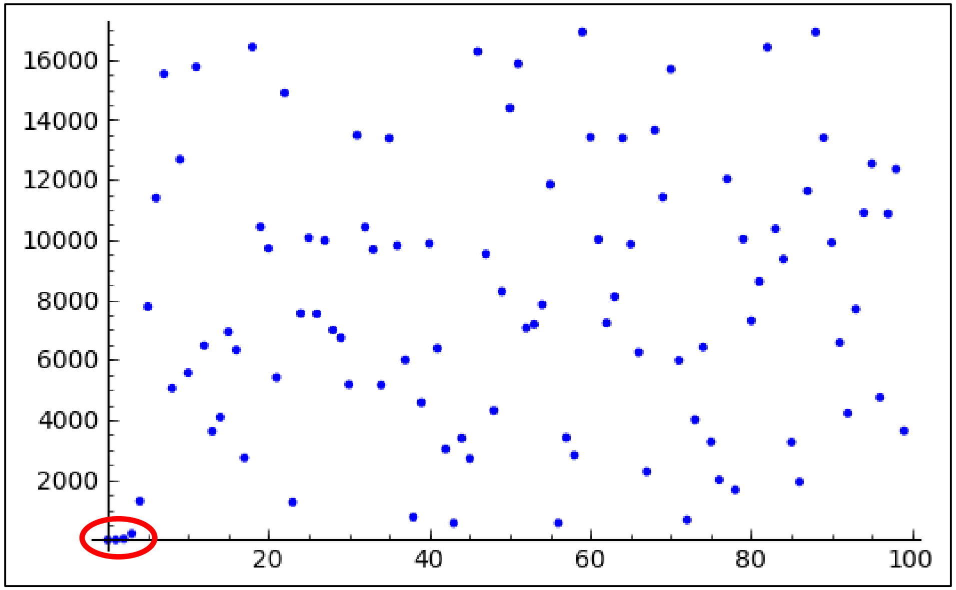
$$g^q = 1 \pmod{p} \text{ AND } g^m \neq 1 \pmod{p} \quad \forall\, m < q$$

  - Cyclic group $\mathrm{G} = < g > = \{1, g, g^2 \ldots g^{q-1}\}$

- DLog problem:
  - Pick $x \in_R \{1, \ldots, q\}$. Compute $X = g^x \pmod{p}$.
  - Given (p, q, g, X) find x.
  - Assumed hard.

EXAMPLES: DLOG, CDH, DDH





- DLog problem:
  - Pick $x \in_R \{1, \ldots, q\}$. Compute $X = g^x \pmod{p}$.
  - Given (p, q, g, X) find x.
  - Assumed hard.

Examples: DLog, CDH, DDH

- DLog problem:
  - Pick $x \in_R \{1, \ldots, q\}$. Compute $X = g^x \pmod p$.
  - Given (p, q, g, X) find x.
  - Assumed hard.
- CDH problem:
  - Pick $x, y \in_R \{1, \ldots, q\}$. Compute $X = g^x \pmod p$; $Y = g^y \pmod p$.
  - Given (p, q, g, X, Y) find g^{xy}.

Just to remind you: $g^{xy} = X^y = Y^x \neq XY = g^{x+y}$

- Solve D-LOG $\Rightarrow$ Solve CDH
- Solve CDH $\nRightarrow$ Solve D-LOG

Examples: DLog, CDH, DDH

- DLog problem:
  - Pick $x \in_R \{1, \ldots, q\}$. Compute $X = g^x \pmod{p}$.
  - Given (p, q, g, X) find x.
- CDH problem:
  - Pick $x, y \in_R \{1, \ldots, q\}$. Compute $X = g^x \pmod{p}$; $Y = g^y \pmod{p}$.
  - Given (p, q, g, X, Y) find g^{xy}.
- DDH problem:
  - Pick $x, y, z \in_R \{1, \ldots, q\}$. Compute X, Y as above
  - Given (p, q, g, X, Y) distinguish g^{xy} from g^z.

How to solve the DLog problem

- In finite fields mod p:
  - Brute force (guess x) – $\mathcal{O}(q)$
  - Baby-step-giant-step: memory/computation tradeoff; $O(\sqrt{q})$
  - Pohlig-Hellman: small factors of q; $O(\log_p q\,(\log q + \sqrt{p}))$
  - Pollard-Rho (+PH): $O(\sqrt{p})$ for biggest factor p of q
  - NFS, Pollard Lambda, …
  - Index Calculus: $\exp((\ln q)^{\frac{1}{3}}(\ln(\ln(q)))^{\frac{2}{3}})$
- Elliptic curves
  - Generic: best case is BSGS/Pollard-Rho
  - Some progress on Index-Calculus attacks recently

Parameter Size vs. Security

ANSSI

Date	Sym.	RSA modulus	DLog Key	DLog Group	EC GF(p)	Hash
<2020	100	2048	200	2048	200	200
<2030	128	2048	200	2048	256	256
>2030	128	3072	200	3072	256	256

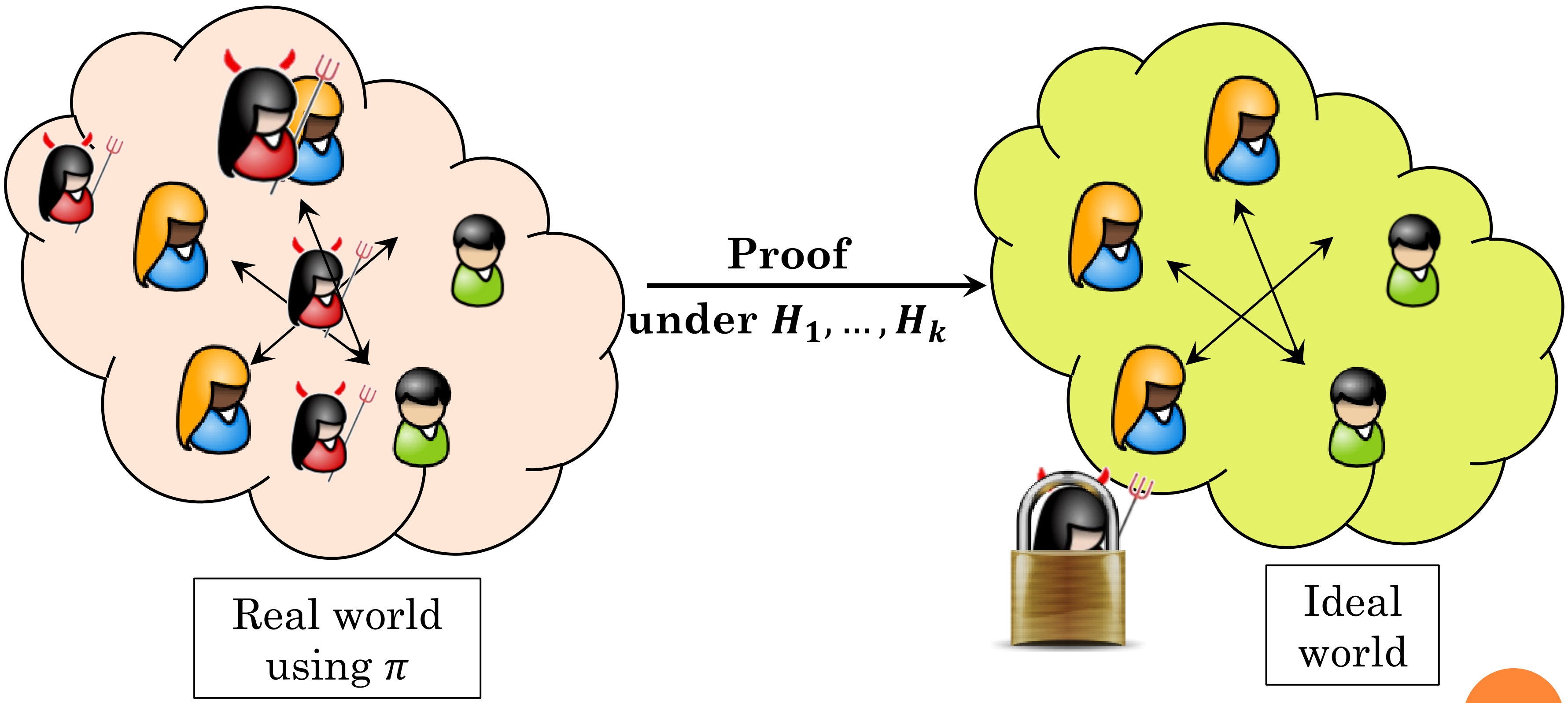
BSI

Date	Sym.	RSA modulus	DLog Key	DLog Group	EC GF(p)	Hash
2015	128	2048	224	2048	224	SHA-224+
2016	128	2048	256	2048	256	SHA-256+
<2021	128	3072	256	3072	256	SHA-256+

Part IV
Security Models

Ideal Provable Security

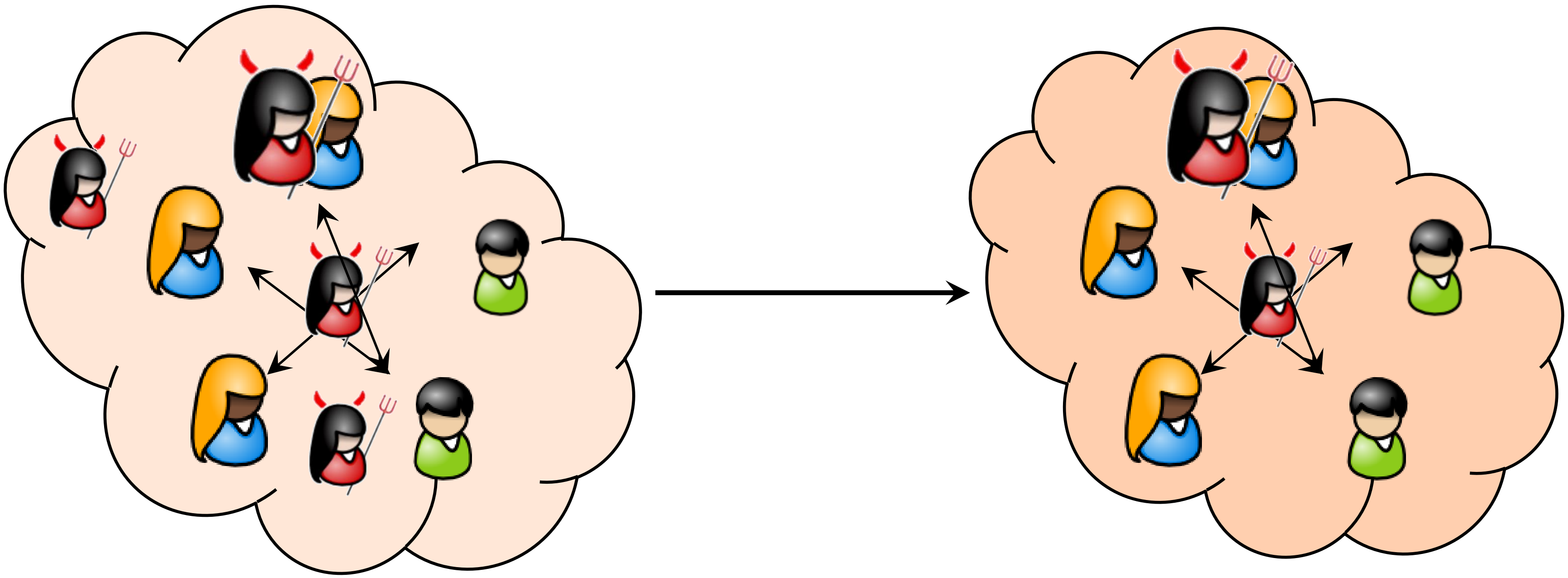
- Given protocol π, assumptions $H_1, \ldots, H_k$



"Real World" is hard to describe mathematically

Provable Security

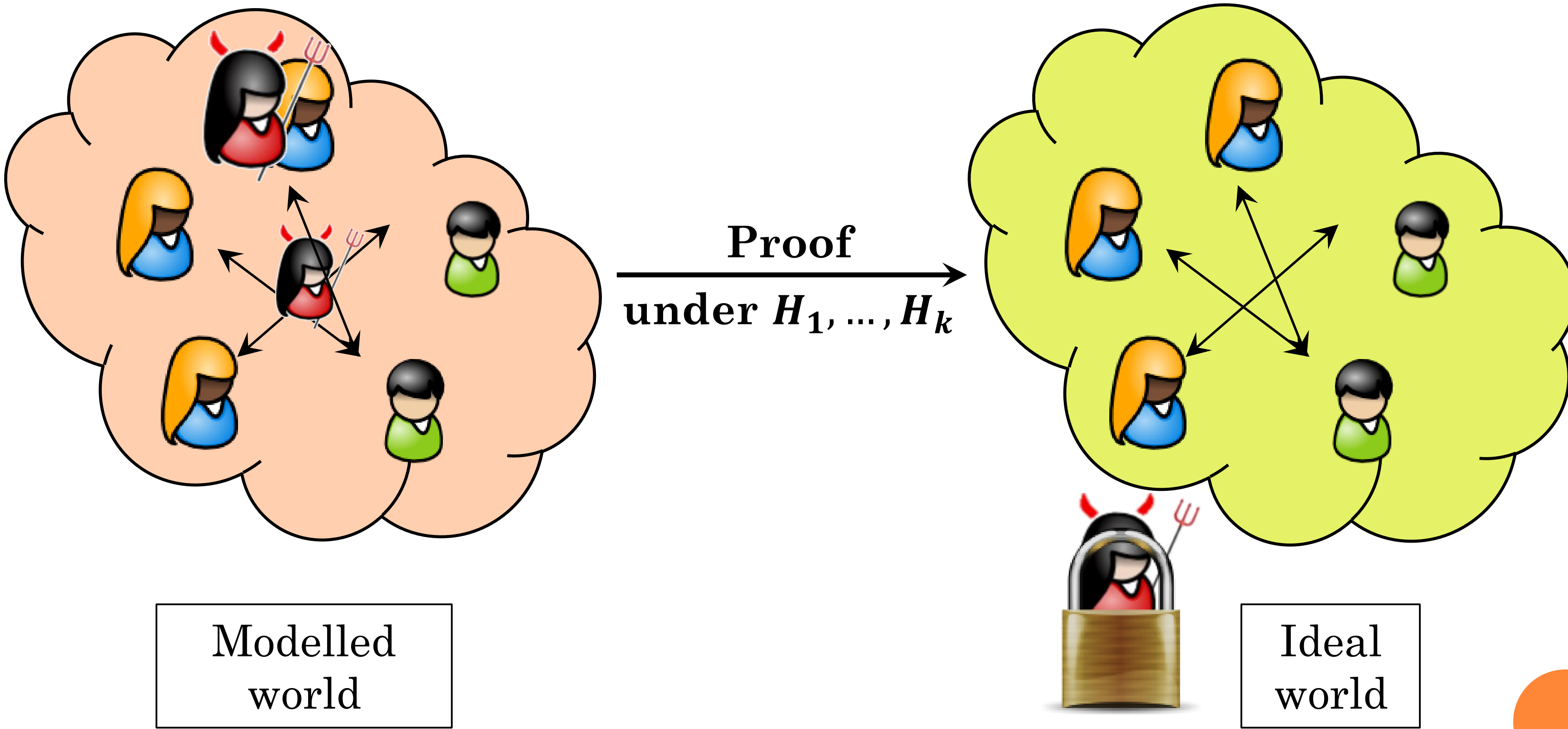
- Two-step process:



Real world using π

Modelled world using π

Provable Security



Components of Security Models

- Adversarial à-priori knowledge & computation:
  - Who is my adversary? (outsider, malicious party, etc.)
  - What does my adversary learn?
- Adversarial interactions (party-party, adversary-party, adversary-adversary – sometimes)
  - What can my adversary learn
  - How can my adversary attack?
- Adversarial goal (forge signature, find key, distinguish Alice from Bob)
  - What does my adversary want to achieve?

GAME-BASED SECURITY

- Participants
  - Adversary A plays a game against a challenger C
    Adversary = attacker(s), has all public information
  - Challenger = all honest parties, has public information and secret information

- Attack
  - Oracles: A makes oracle queries to C to learn information
  - Test: special query by A to C to which A responds
    sometimes followed by more oracle queries
  - Win/Lose: a bit output by C at the end of the game

Measuring Adversarial Success

- Winning a game; winning condition:
  - Depends on relation R on $(*, < \text{game} >)$, with $< \text{game} > =$ full game input (of honest parties and A)
  - Finally, A outputs x, wins if $(x, < \text{game} >) \in R$
- Success probability:
  - What is the probability that A "wins" the game?
  - What is the probability measured over? (e.g. randomness in $< \text{game} >$, sometimes probability space for keys, etc.)
- Advantage of Adversary:
  - How much better is A than a trivial adversary?

Adversarial Advantage

- Forgery type games:
  - A has to output a string of a "longer" size
  - Best trivial attacks: guess the string or guess the key
  - Advantage:

    $\mathrm{Adv}[A] = \mathrm{Prob}[A \text{ wins the game}]$
- Distinguishability-type games:
  - A must distinguish between 2 things: left/right, real/random
  - Best trivial attacks: guess the bit (probability $^1/_2$)
  - Advantage (different ways of writing it):

    $\mathrm{Adv}[A] = \mathrm{Prob}[A \text{ wins the game}] - {}^1/_2$

    $\mathrm{Adv}[A] = 2\,|\,\mathrm{Prob}[A \text{ wins the game}] - {}^1/_2\,|$

Security Models – Conclusions

- Requirements:
  - Realistic models: capture "reality" well, making proofs meaningful
  - Precise definitions: allow quantification/classification of attacks, performance comparisons for schemes, generic protocol-construction statements
  - Exact models: require subtlety and finesse in definitions, in order to formalize slight relaxations of standard definitions

- Provable security is an art, balancing strong security requirements and security from minimal assumptions

Example: Pseudorandomness

- Perfect confidentiality exists:
  - Given by the One-Time Pad

$$c \coloneqq m \oplus k$$

  - XOR operation hides plaintext m entirely

- Disadvantages:
  - Need long keys (as long as plaintext)
  - Have to generate them at every encryption

- Generating long randomness:
  - Use a pseudorandom generator!

PRGs

- Principle:
  - Start from small, truly random bitstring s, generate large pseudorandom strings

$$\text{PRG}: \{0,1\}^m \rightarrow \{0,1\}^n, \qquad \text{for } m \ll n$$

- Security (intuitive):
  - The adversary gets to see many output strings
  - In practice: PRGs used for randomness in encryption, signature schemes, key-exchange…
  - Adversary's goals (examples):
    - Predict next/former random number
    - "Cancel out" randomness

Secure PRGs

- Ideally PRG output should look "random"
- Formally:
  - Allow A to see either truly random or PRG output
  - The adversary wins if it distinguishes them

- Security game:
  - Challenger picks seed of generator (A does not get it)
  - Challenger chooses a secret bit b
  - A can request random values
    - If $b = 1$ then Challenger returns $x \overset{\$}{\leftarrow} \{0,1\}^n$
    - If $b = 0$ then Challenger returns $x \leftarrow \mathrm{PRG}(s)$
  - A must output a guess bit d
  - Winning condition: A wins iff. $d = b$

The security definition

- $s \xleftarrow{\$} \{0,1\}^m$

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

  $d \leftarrow A^{Gen_b()}(m,n)$

  A wins iff $d = b$

$\mathrm{Gen}_b()$

- If $b = 1$, return $x \xleftarrow{\$} \{0,1\}^n$
- Else, return $x \leftarrow \mathrm{PRG}(s)$

- Success probability is at least ½ . Why ?

- **<u>(k, ϵ)-secure PRG</u>**:

  A pseudorandom generator PRG is (k, ϵ)-secure if, and only if, an adversary making at most k queries to Gen_b wins w.p. at most $\frac{1}{2} + \epsilon$

Part V
Proofs of Security

PROOFS BY REDUCTION

- Say we have a primitive P
- We make assumptions S_1, S_2

- **Goal**: prove that if S_1, S_2 hold, then P is secure
- **Statement**: If there exists an adversary A against P, then there exists adversaries B_1, B_2 against assumptions S_1, S_2, such that:

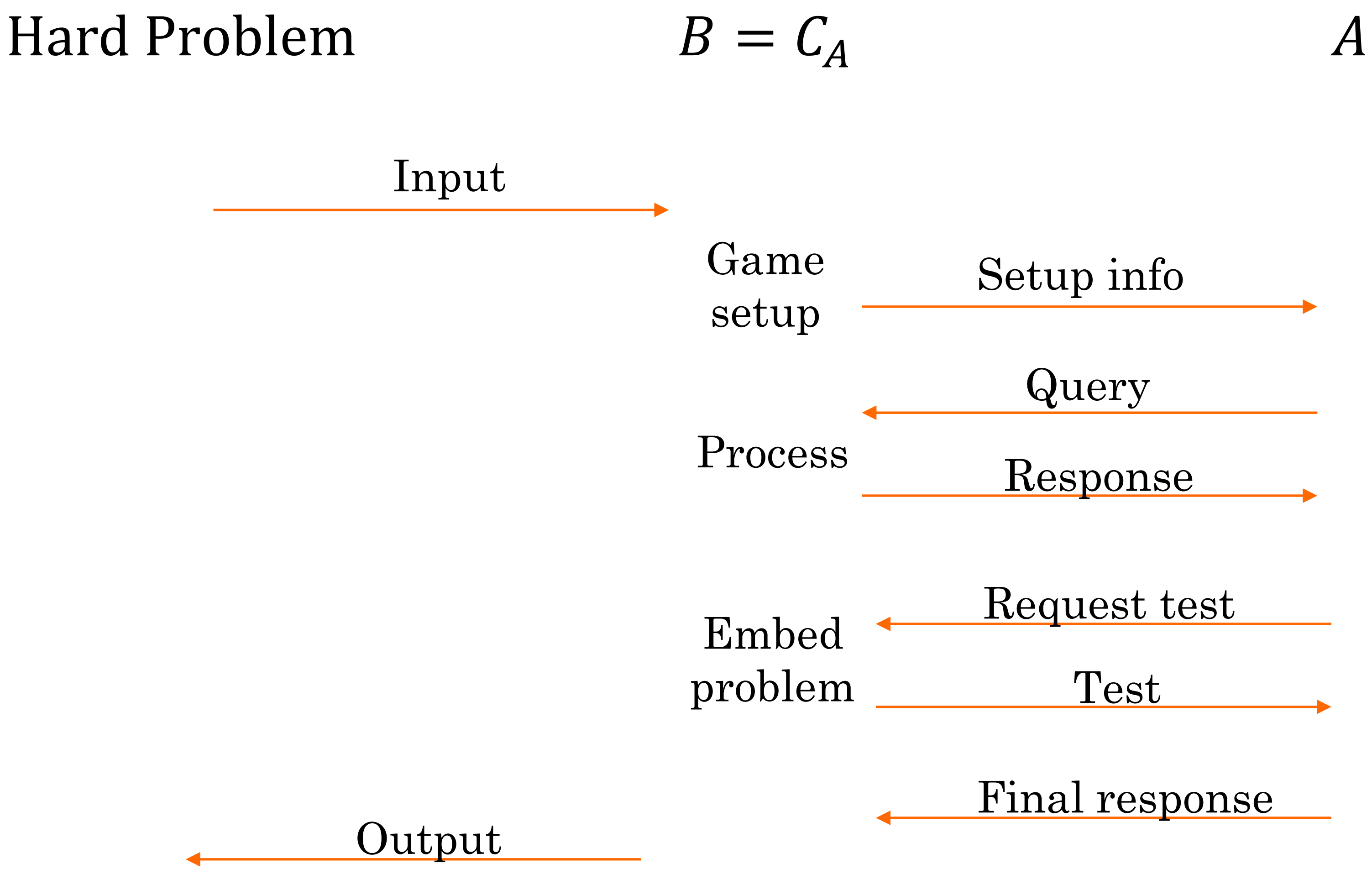
$$\mathrm{Adv}(A) \leq f(\mathrm{Adv}(B_1), \mathrm{Adv}(B_2))$$

- Idea: if $\mathrm{Adv}(A)$ is significant, then so is at least one of $\mathrm{Adv}(B_1), \mathrm{Adv}(B_2)$, breaking at least one assumption

Reducing security to hard problem

- Designed primitive has some game-based definition
  - A gets to query a challenger C
  - C gets to set up the system
  - There is a test phase
  - A will eventually answer the test and win/lose
- Hard problem of the form: given Input, find Output
- Strategy: use A to construct solver B for hard problem
  - B gets Input
  - B uses Input to run A on some instance of A's game
  - Finally, B receives A's answer to its test
  - B processes A's response into some Output

Reductions



Hard Problem
$B = C_A$
A
Input
Game setup
Setup info
Query
Process
Response
Request test
Embed problem
Test
Final response
Output

Constructing a Reduction

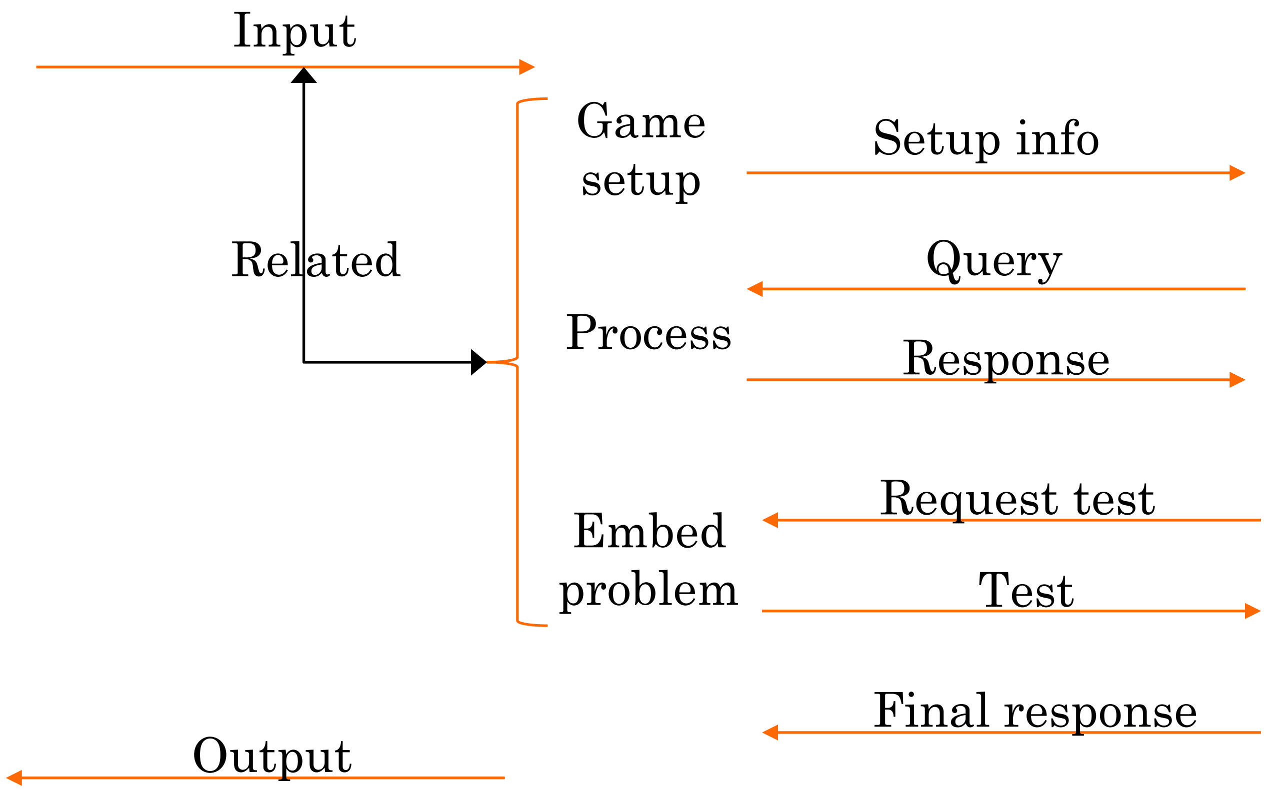
- A acts as a black-box algorithm (we don't know how it works in order to win its game)

- B can send to A whatever it wants. However:
  - We want to bound B's winning probability on A's
  - But, A can only win if the game input is coherent
  - So B must simulate coherent input/output to A's queries

  - Also, B must ultimately solve a hard problem
  - To produce correct output, A's test response must give B the correct output with very high probability

Reductions

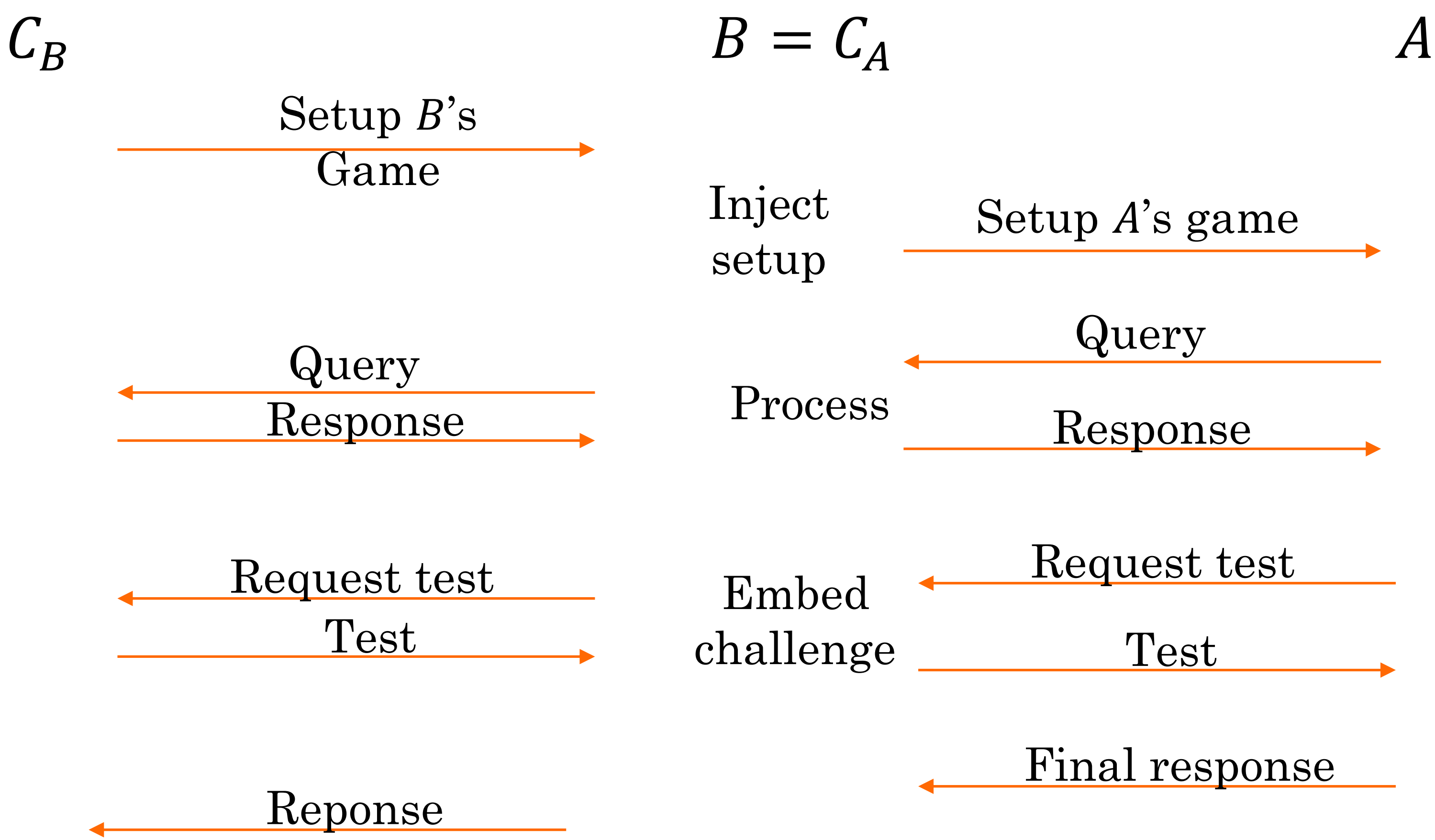
Hard Problem $\quad B = C_A \quad A$



Reduction to security of component

- Designed primitive has some game-based definition
  - A gets to query a challenger C
  - C gets to set up the system
  - There is a test phase
  - A will eventually answer the test and win/lose
- Component also has game-based definition
- Strategy: use A to construct solver B for hard problem
  - B gets Setup info and can query its challenger
  - B embeds its game in some instance of A's game
  - Finally, B receives A's answer to its test
  - B processes A's response into a test response of its own

Reductions


C_B
$B = C_A$
A
Setup B's Game
Inject setup
Setup A's game
Query
Response
Process
Query
Response
Request test
Test
Embed challenge
Request test
Test
Final response
Reponse

Example: Bigger PRG

- Say we have a secure pseudorandom generator:

$$G_{\text{small}}: \{0,1\}^m \rightarrow \{0,1\}^n$$

- We want to construct a bigger PRG:

$$G_{\text{big}}: \{0,1\}^m \rightarrow \{0,1\}^{2n}$$

- Instantiating G_{big}:
  - Setup: choose $s \xleftarrow{\$} \{0,1\}^m$
  - Evaluation: $G_{\text{big}}(s) \coloneqq G_{\text{small}}(s) \mid G_{\text{small}}(s)$

Claim: If G_{small} is secure, then so is G_{big}

Security of our design

- **Statement**: For any $(k, \epsilon_{\text{big}})$-adversary A against the security of G_{big}, there exists a $(2k, \epsilon_{\text{small}})$-adversary B against the security of G_{small} such that:

$$\epsilon_{\text{big}} \leq \epsilon_{\text{small}}$$

- Both adversaries play the same game:
  - $s \xleftarrow{\$} \{0,1\}^m$

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

$$d \leftarrow A^{Gen_b()}(m, n)$$

$$\overline{}$$

A wins iff $d = b$

$\text{Gen}_b()$

- If $b = 1$, return $x \xleftarrow{\$} \{0,1\}^n$
- Else, return $x \leftarrow \text{PRG}(s)$

Constructing the Reduction

C_B $\qquad B = C_A$ $\qquad A$

$s \xleftarrow{\$} \{0,1\}^m$

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

Setup done

Setup done

Query

Query Gen_b

- If $b = 1$, return $x \xleftarrow{\$} \{0,1\}^n$
- Else, return $x \leftarrow \mathrm{G}_{\mathrm{small}}(s)$

x_1

Query Gen_b

x_2

$x_1 \mid x_2$

Guess bit d

Output d

Analysis of the Reduction

- Number of queries:
  - For each query, A expects a $2n$ response, whereas A only gets n bits from its challenger
  - Thus B needs twice as many queries as A

- Accuracy of B's simulation of C_A
  - In A's game if $b = 1$, A gets $2n$ truly random bits
  - And if $b = 0$, it expects $G_{\text{small}}(s) \mid G_{\text{small}}(s)$
  - B queries its own challenger for output
    - If C_B drew bit $b = 1$, it outputs n truly random bits
    - Else, it outputs $G_{\text{small}}(s)$
  - The simulation is perfect: $\Pr[B \text{ wins}] = \Pr[A \text{ wins}]$

EXERCISES

- Why does this proof fail if we have two secure PRGs: $G_1, G_2 : \{0,1\}^m \rightarrow \{0,1\}^n$ and we construct $G : \{0,1\}^m \rightarrow \{0,1\}^{2n}$ as follows:
  - Setup: choose $s \overset{\$}{\leftarrow} \{0,1\}^m$
  - Evaluation: $G(s) \coloneqq G_1(s) \mid G_2(s)$

- Will the proof work if $G(s) \coloneqq G_1(s) \oplus G_2(s)$?