Complex Primitives and Protocols

Reductions, Authentication, Simulation-based models

From previous lecture

- Pairings
  - Powerful tool, based on bilinearity
  - Not always easily computable
  - Makes solving DDH trivial; new assumptions: CBDH, DBDH
- Identity-Based Encryption (IBE)
  - Encrypt using the public identity of the receiver
  - Require global setup to deal with secret key generation
  - Boneh/Franklin: IND-CPA using pairings, in the ROM
  - IND-CCA security based on IBE
- Signature Schemes
  - Signatures in the standard model: BSW

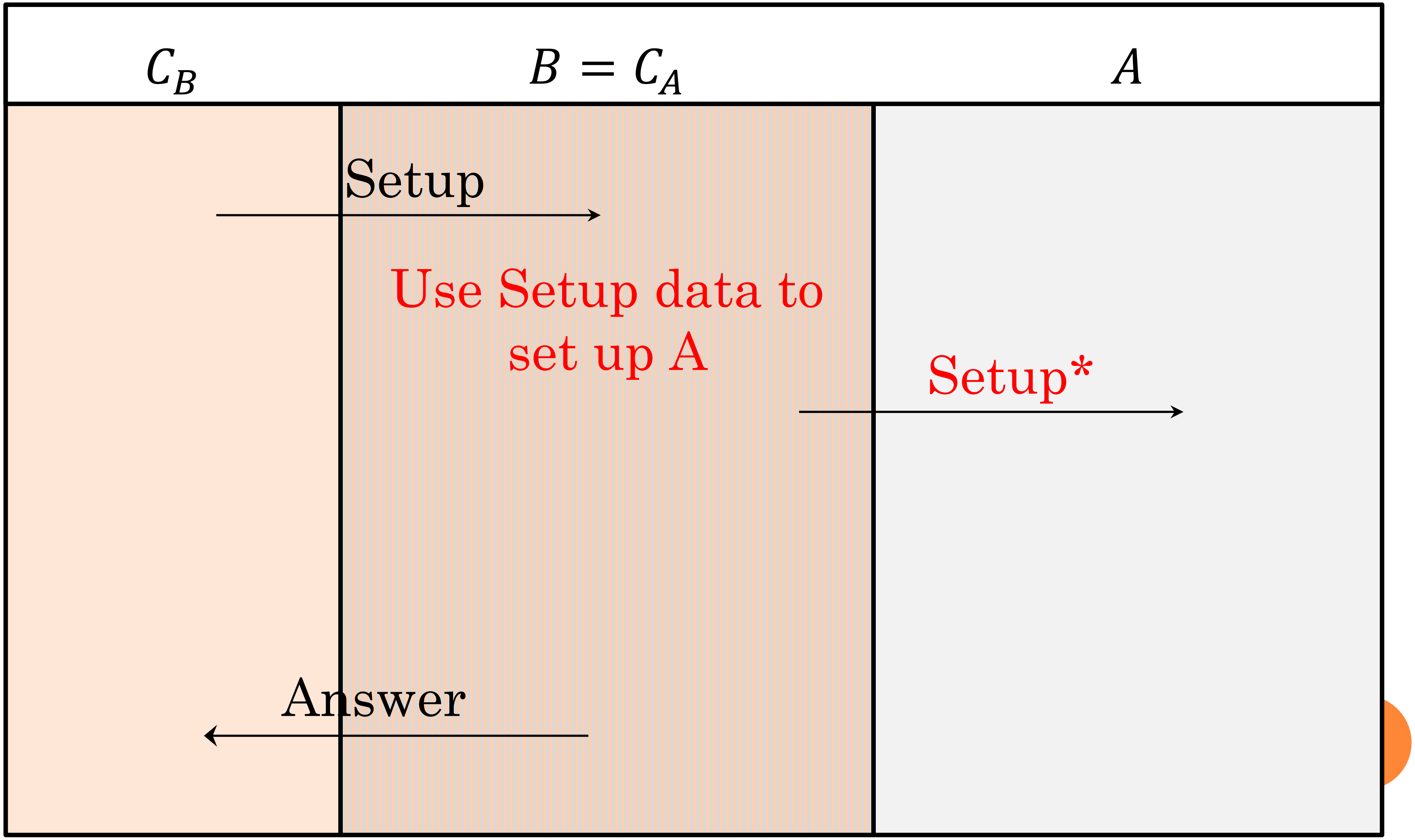
Part I
Recap: Reductions

Understanding a Reduction

- Typical statement to prove:
  - If there exists adversary A winning game $\mathbb{G}_A$ with probability p_A, then there exists adversary B winning game $\mathbb{G}_B$ with probability $p_B = f(p_A)$.

- What happens in the proof
  - We need to keep track of two games.
  - In the main game (the one we need to analyse later), B plays against a challenger C_B
  - In order to win its game, B must use adversary A
  - But adversary A is like a machine:
    - It only works if given the right information
    - Hence, B must provide that information

Reductions



Simulating A's game

- Why does B not simply play the game honestly?
- After all, if B plays A's challenger as the challenger does, we are guaranteed that A works as usual!

Simulating A's game

- Why does B not simply play the game honestly?
- After all, if B plays A's challenger as the challenger does, we are guaranteed that A works as usual!

- Yes… but…
- Remember B plays his own game. He wants to win his own game, and so he has to learn something about the secrets that C_B generated

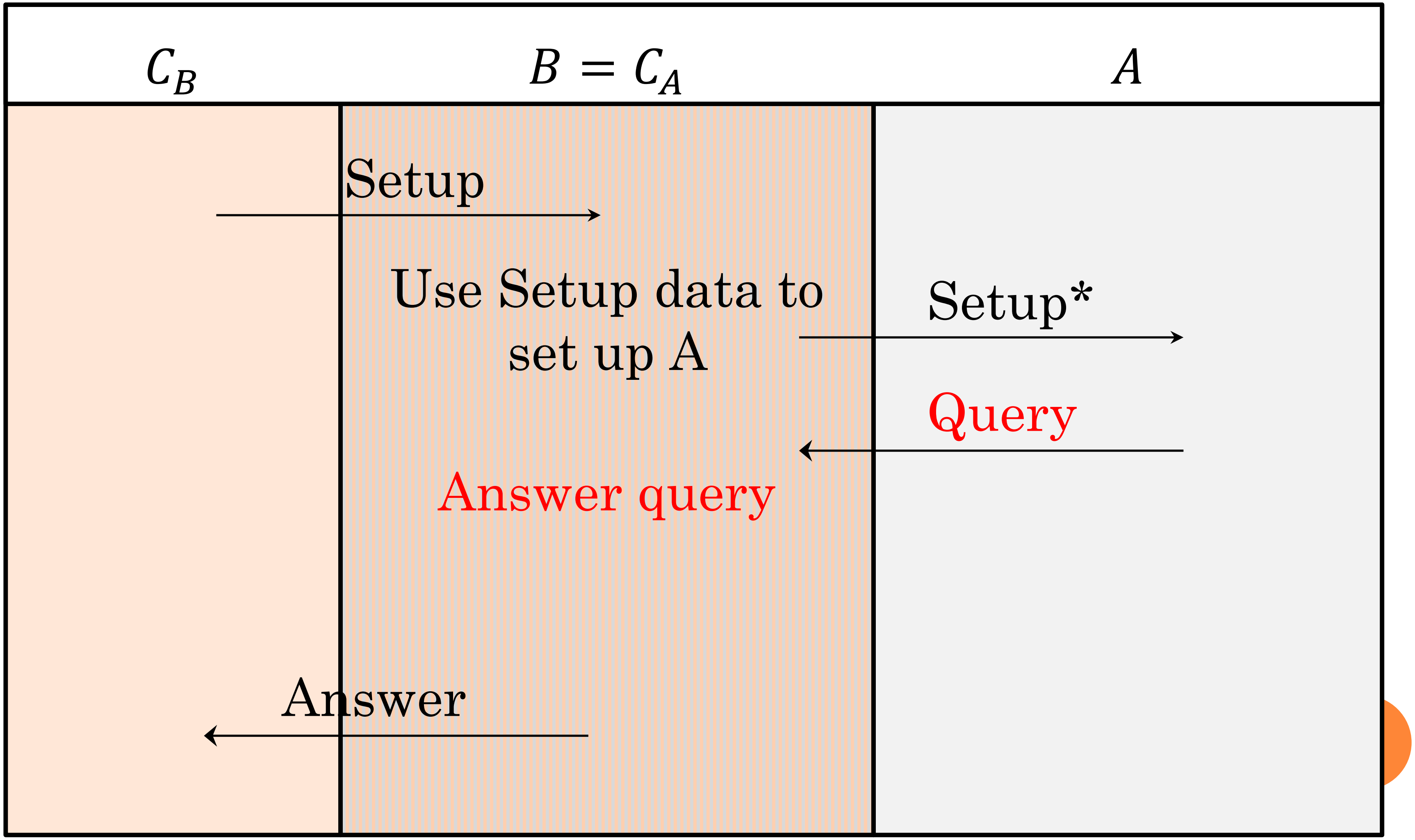
- The only way B can do this is by injecting parts of his own setup or oracle output into A's queries

Simulating A's game

- Setup:
  - B must set up the game for A
  - B may use some of its own setup information, like a public key, or it may not

- Queries:
  - For each of A's oracle queries, B must answer the query as A's challenger would
  - If B answers badly, A may, or may not realise it
  - However, if B's answer is statistically wrong for the query, this influences A's success probability
  - Our goal: simulating A's queries as well as possible

Reductions



C_B

$B = C_A$

A

Setup

Use Setup data to set up A

Setup*

Query

Answer query

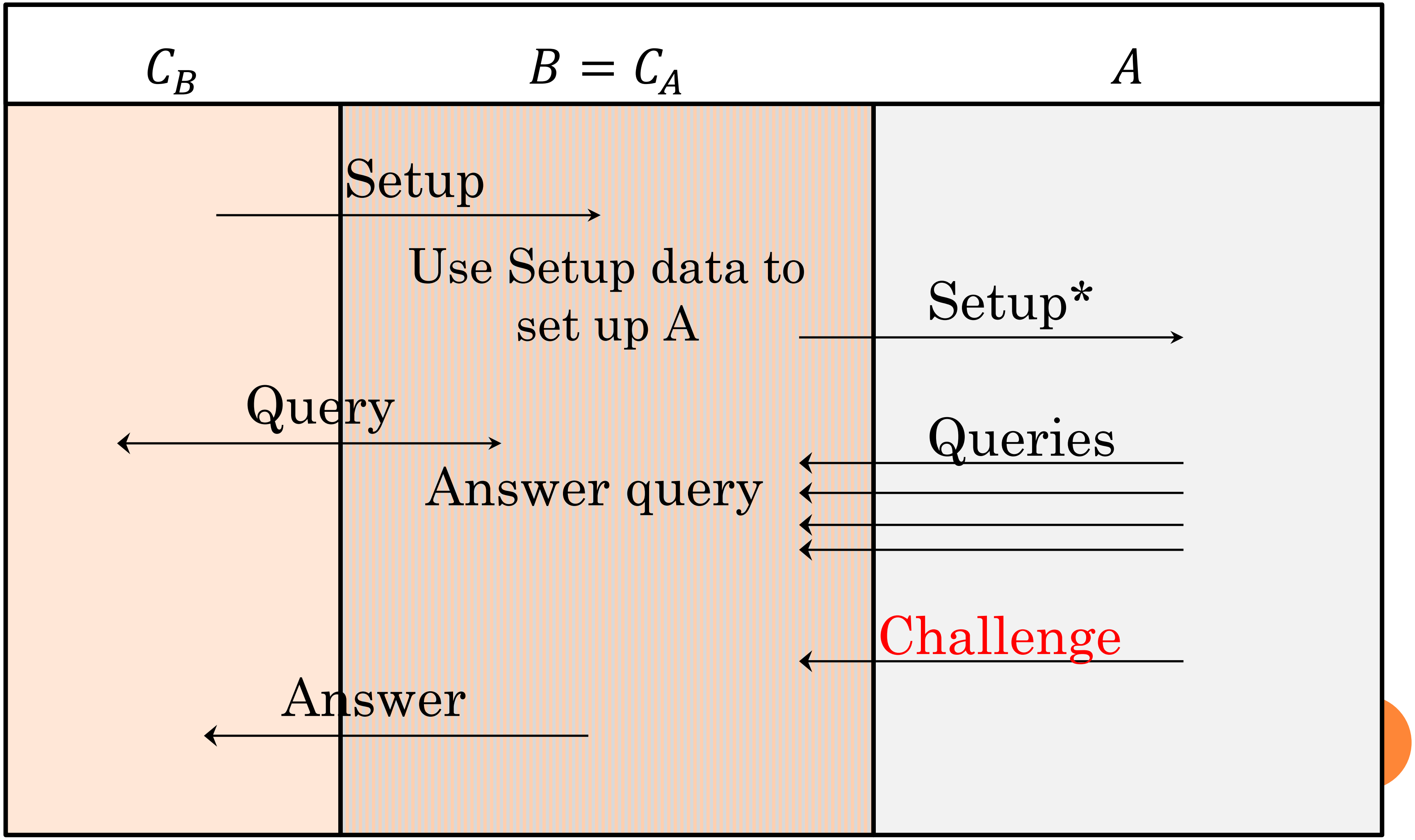
Answer

Answering Queries

- How does B answer A’s queries?
  - Option 1: B acts as an honest challenger – the output is perfectly simulated for A
  - Option 2: B uses its own oracle queries to C_B in order to answer to A
  - Option 3: B makes up the answer itself, usually injecting part of the challenge

- In general – and it’s not a rule:
  - We use option 1 whenever B answers queries not relevant to A’s challenge query
  - We use option 2 in real-from-random indistinguishability (for A and/or B)
  - We use option 3 when we still have parts of B’s setup that were not injected into the game

Reductions



C_B

$B = C_A$

A

Setup

Use Setup data to set up A

Setup*

Query

Queries

Answer query

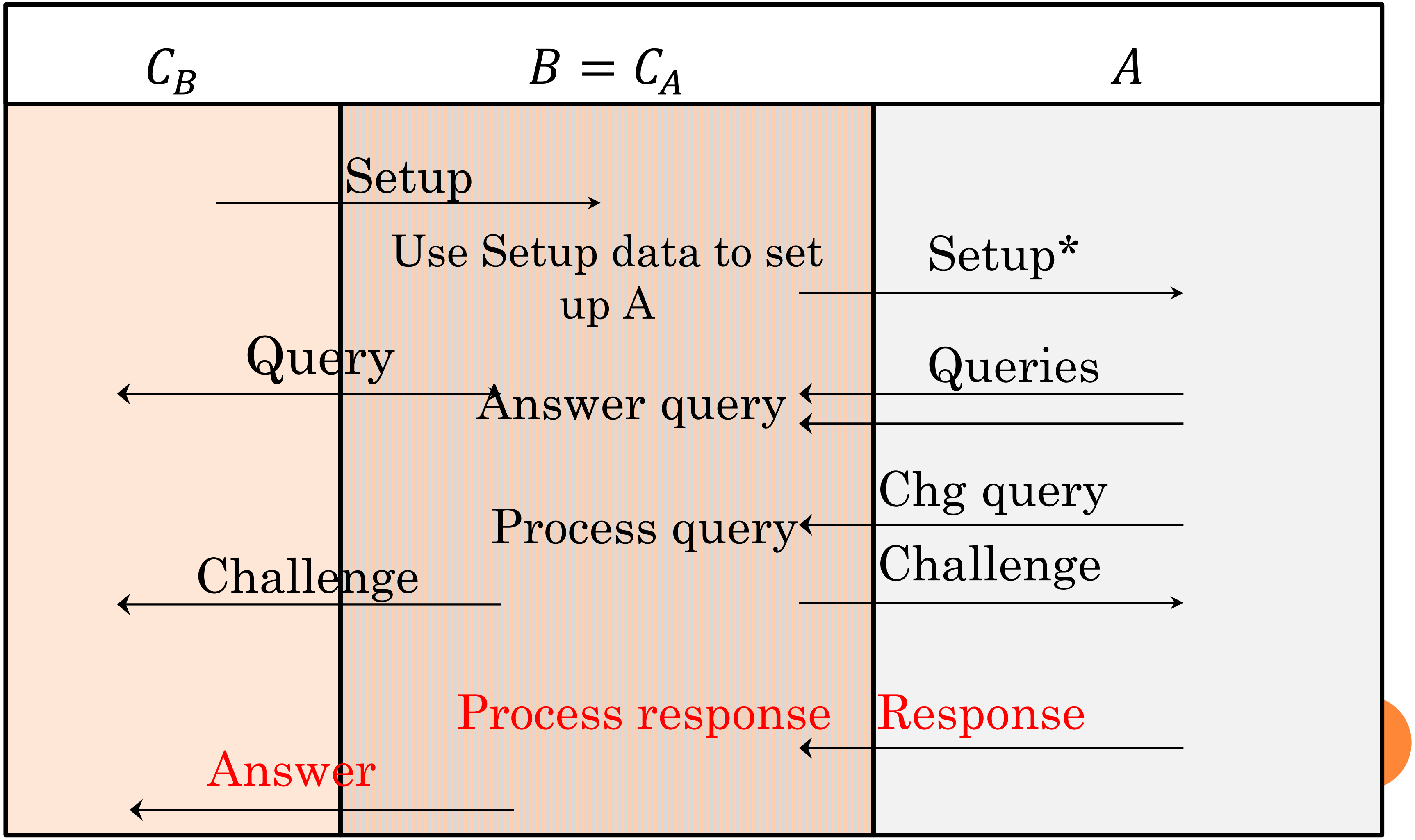
Challenge

Answer

Answering Challenge Query

- A makes a special challenge query in its game
  - B has to answer that query, simulating A's challenger
  - Option 1: Use A's challenge to generate B's own challenge
  - Option 2: Inject B's Setup information into the response
  - Option 3: Generate challenge honestly towards receiving and using the returned response
- A might make more oracle queries
  - B might use a different oracle response strategy after chg.
- Finally, A sends B a response to its challenge
  - B needs to use this response to answer its challenger's challenge

Reductions



C_B

$B = C_A$

A

Setup

Use Setup data to set up A

Setup*

Query

Queries

Answer query

Chg query

Process query

Challenge

Challenge

Process response

Response

Answer

Analysis

- After describing reduction (adversary B), we need to analyse its success probability

- Note: adversary A only succeeds with its usual probability if B's simulation is statistically flawless
  - Otherwise, A only wins with trivial probability

- Probability usually computed based on conditional probabilities. We condition on an event E (for instance, that C_B chooses a bit = 1). Then:

$$\mathbb{P}[B \text{ wins } \mathbb{G}_B] = \mathbb{P}[B \text{ wins } \mathbb{G}_B \mid E]\ \mathbb{P}[E] + \mathbb{P}[B\ wins\ \mathbb{G}_B \mid \neg E]\ \mathbb{P}[\neg E]$$

Part II

Authentication

Authentication/Identification

- Setting, 2 parties :
  - Prover: party associated with some private parameters that are user-specific
  - Verifier: party that is entitled to verify the legitimacy of provers
- Goal: want provers to be authenticated as legitimate by the verifier
- Example:
  - Passports for travelling, KorriGo card
  - Unlocking your car, parking meters
  - Username/Password, PIN codes – banking/ phones

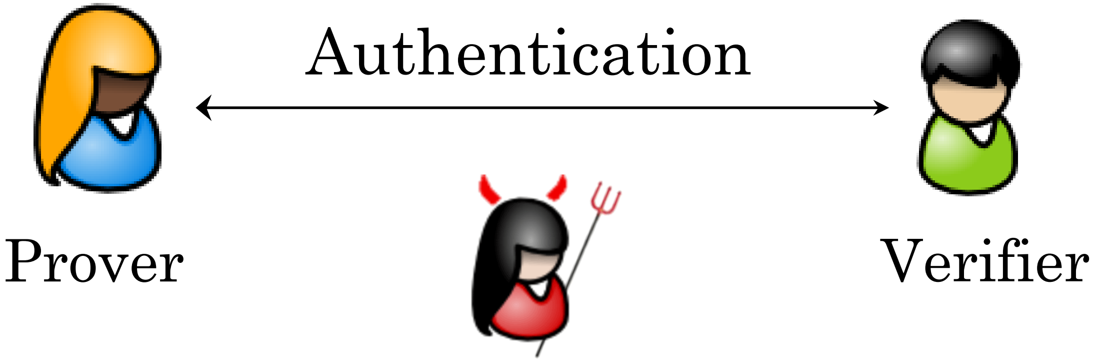
Authentication vs identification

- A passport identifies
- A badge (one of many valid ones) authenticates

- Authentication: verifier can establish that the prover is legitimate, but it does not learn the identity of that prover. Output is an accept/reject bit

- Identification: verifier can establish the prover's identity, amidst several legitimate provers. Output is identity *ID*

AUTHENTICATION



- Functionality/Correctness:
  - Legitimate prover can always authenticate to the verifier
- Security:
  - No illegitimate prover can authenticate to the verifier

The primitive

- Authentication can be either symmetric or public-key, depending on infrastructure and resources

- Symmetric-key authentication:
  - $\mathrm{KGen}(1^{\lambda})$: on input a security parameter, outputs a symmetric key $\mathrm{s}k$ to both parties
  - $\mathrm{Prove}(sk)$: on input the secret key this algorithm runs the prover's part of the algorithm
  - $\mathrm{Verify}(sk)$: on input the secret key, this algorithm runs the verifier's part of the algorithm

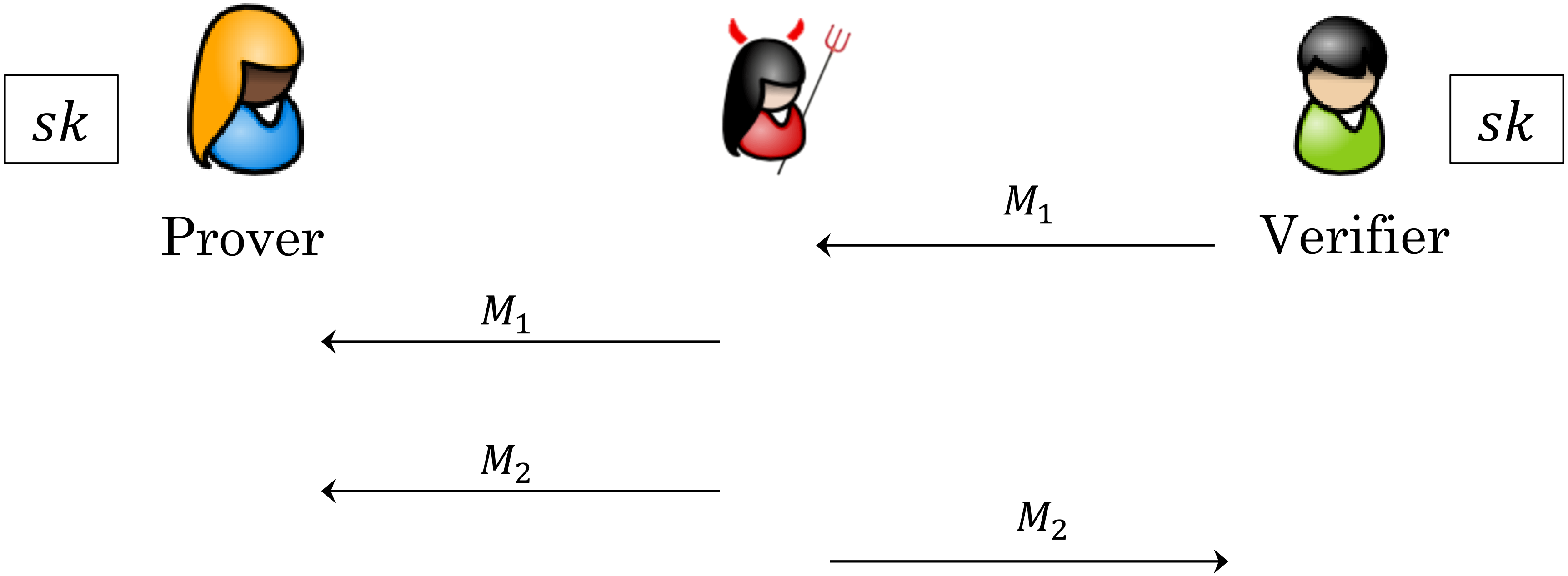
  - Algorithms $(\mathrm{Prove}, \mathrm{Verify})$ are run together as a tuple

Security in Authentication

- Correctness: $\forall\ sk \leftarrow \mathrm{KGen}(1^{\lambda})$, running Prove and Verify together on input sk yields 1

- Security:
  - The adversary is a Man-in-the-Middle
  - It can communicate with the Prover and the Verifier
  - Its goal is to be authenticated by the Verifier
  - Let's talk about trivial attacks

Relay Attacks



sk Prover

sk Verifier

M_1

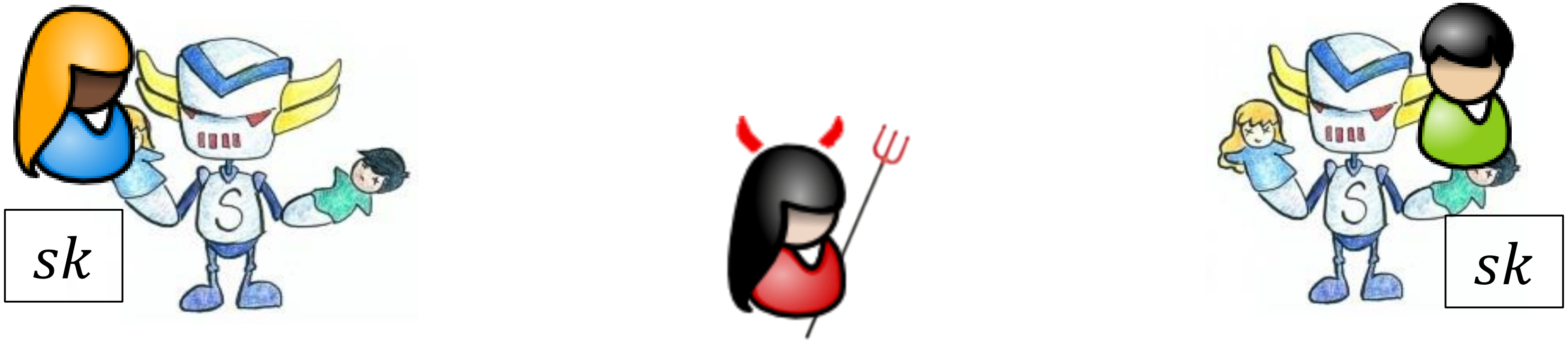
M_1

M_2

M_2

- Relay attacks bypass any kind of cryptography: encryption, hashing, signatures, etc.
- Countermeasure: distance bounding protocols (not covered in tis course)

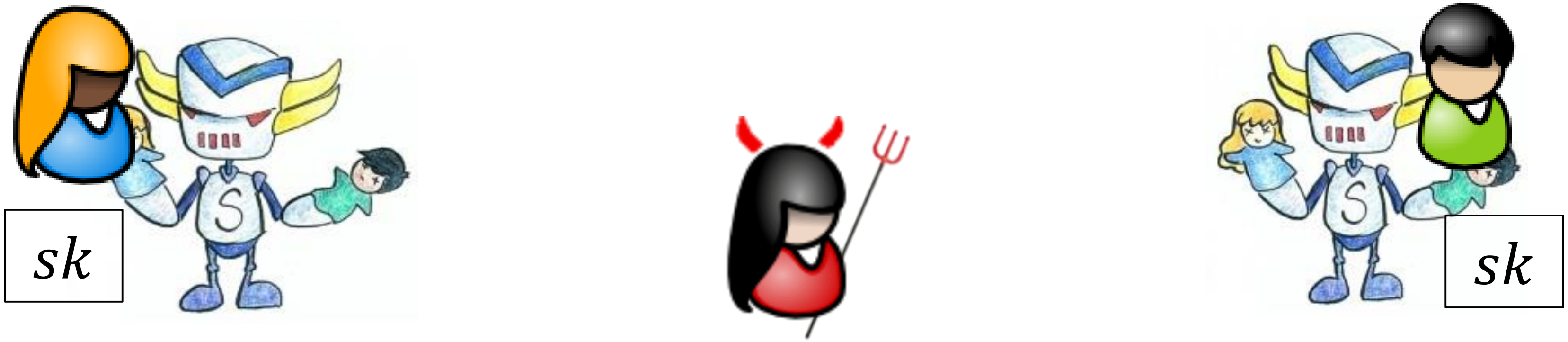
FORMALIZING THE SECURITY MODEL



Prover Verifier

- Challenger first generates the secret key (and keeps it)
- Protocol is run in sessions with two parties (one plays the role of the prover, the other the role of the verifier)
  - Prover-adversary OR adversary-verifier
- Adversary can send messages in prover-adv. or adv.-verifier sessions

Formalizing the Model



Prover

Verifier

- Challenger first generates the secret key (and keeps it)
- Adversary can initiate sessions (P-A or A-V)
  - If A wants to see honest prover-verifier sessions, it can open a prover-adversary session and an adversary-verifier session and then relay communication between the sessions
- Winning: A succeeds in A-V session, without relaying

ORACLES

- Adversary can initiate sessions (P-A or A-V)
  - If A wants to see honest prover-verifier sessions, it can open a prover-adversary session and an adversary-verifier session and then relay communication between the sessions
- Oracle:
  - NewSession($*$): input either P_1 = Prover or P_2 = Verifier outputs session "handle" π
- Session handles:
  - In practice a unique index, like a number, which serves to identify one session for another
  - We call it handle because we use it to handle the session

More oracles

- Say we create a new session with the prover.
  - $\pi_1 \leftarrow \text{NewSession}(P_1)$

- We want A to communicate with the prover in session π_1
  - It will not be the prover answering, but the challenger
  - Need a second oracle for sending messages

- Oracle:
  - Send($*$,$*$): input handle π and message $m \in M \cup \{\text{Prompt}\}$

    transmits m to partner in π, outputs m'

- Send(π_1, Prompt): if prover starts the protocol, output the first protocol message; else, output $\perp$

More Oracles

- Now A can initiate sessions and talk to its partner
- Every time A sends a message to its partner, the other party responds with a message (or with \bot in case of aborts or errors)
- In an A-V session, V computes authentication/ rejection bit
  - This bit is not always sent in clear
  - But we can give A an oracle to find out what it is
- Oracle:
  - Result($*$): on input a handle π with partner P_2, output 1 if P_2 accepted partner in π, 0 if partner is rejected, and $\perp$ otherwise

Oracles for Impersonation

- Oracles:
  - NewSession($*$): input either P_1 = Prover or P_2 = Verifier
    outputs session "handle" π
  - Send($*$,$*$): input handle π and message $m \in M \cup \{\text{Prompt}\}$
    transmits m to partner in π, outputs m'
  - Result($*$): on input a handle π with partner P_2, output
    1 if P_2 accepted partner in π, 0 if partner
    is rejected, and $\perp$ otherwise

Parameters and Winning

- Parameters:
  - Number of sessions with prover: q_P
  - Number of sessions with verifier: q_V
  - Max number of prover-verifier sessions A can run: $\min\{q_p, q_V\}$
- Winning:
  - The result of at least one A-V session is 1
  - And for that session there is no relaying to a P-A session
  - No relaying: just compare session transcripts

Security Game

- Impersonation Security:

  $sk \leftarrow \text{KGen}(1^\lambda)$

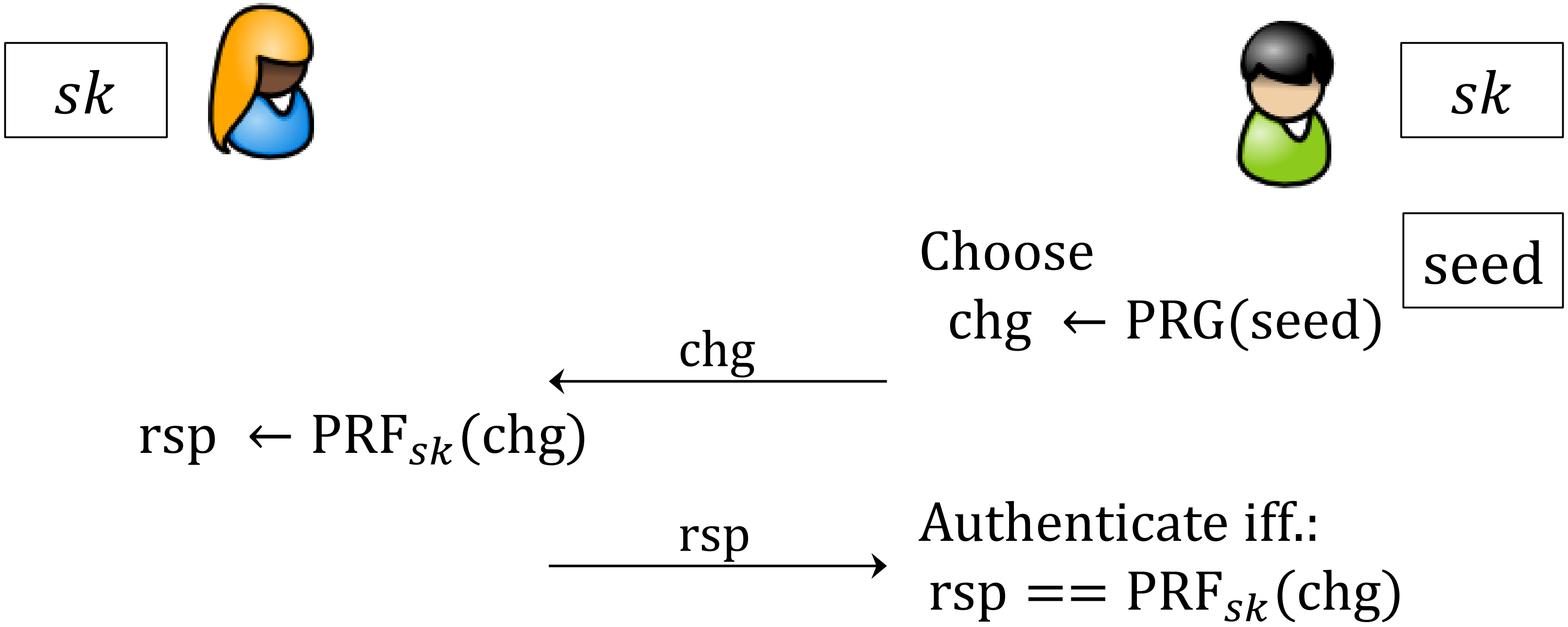
  $\text{Stop} \leftarrow A^{\text{NewSession}(*).\text{Send}(*,*),\text{Result}(*)}(1^\lambda)$

  A wins iff.: $\exists\ \pi^* \leftarrow \text{NewSession}(P_2)$ such that:

  $\text{Result}(\pi^*) = 1$ and $\forall\ \pi \leftarrow \text{NewSession}\ (P_1)$ s.t. the query is made between the creation of π^* and its termination (last message) it holds: $\text{Transcript}(\pi^*) \neq \text{Transcript}(\pi)$

- Parametrization:
  - Protocol is (q_P, q_V, ϵ)-impersonation secure iff. $\forall\ A$ with at most q_P $\text{NewSession}(P_1)$ queries and at most q_V queries to *Newsession* (P_2) wins w.p. at most ϵ.

An Authentication Protocol



sk

sk

seed

Choose
$\text{chg} \leftarrow \text{PRG}(\text{seed})$

chg

$\text{rsp} \leftarrow \text{PRF}_{sk}(\text{chg})$

rsp

Authenticate iff.:
$\text{rsp} == \text{PRF}_{sk}(\text{chg})$

- The seed is only generated once
- Take $\text{PRG}: \{0,1\}^{|\text{seed}|} \rightarrow \{0,1\}^n$ $\text{PRF}: \{0,1\}^{|sk|} \times \{0,1\}^n \rightarrow \{0,1\}^m$

Impersonation Security Theorem

- Theorem:
  - For any (q_P, q_V, ϵ_A)-adversary A against the impersonation security of the Challenge-Response protocol…
  - … There exist adversaries B against the pseudorandomness of PRG making at most q_V PRG queries and winning with probability $\frac{1}{2} + \epsilon_{\text{PRG}}$; and C against the pseudorandomness of PRF winning with probability $\frac{1}{2} + \epsilon_{\text{PRF}}$ such that:

$$\epsilon_A \leq \epsilon_{\text{PRG}} + \binom{q_V}{2} 2^{-|\text{chg}|} + \epsilon_{\text{PRF}} + q_P\left(2^{-|\text{chg}|} + 2^{-|\text{rsp}|}\right)$$

- Proof: by game hopping

Game hop #1

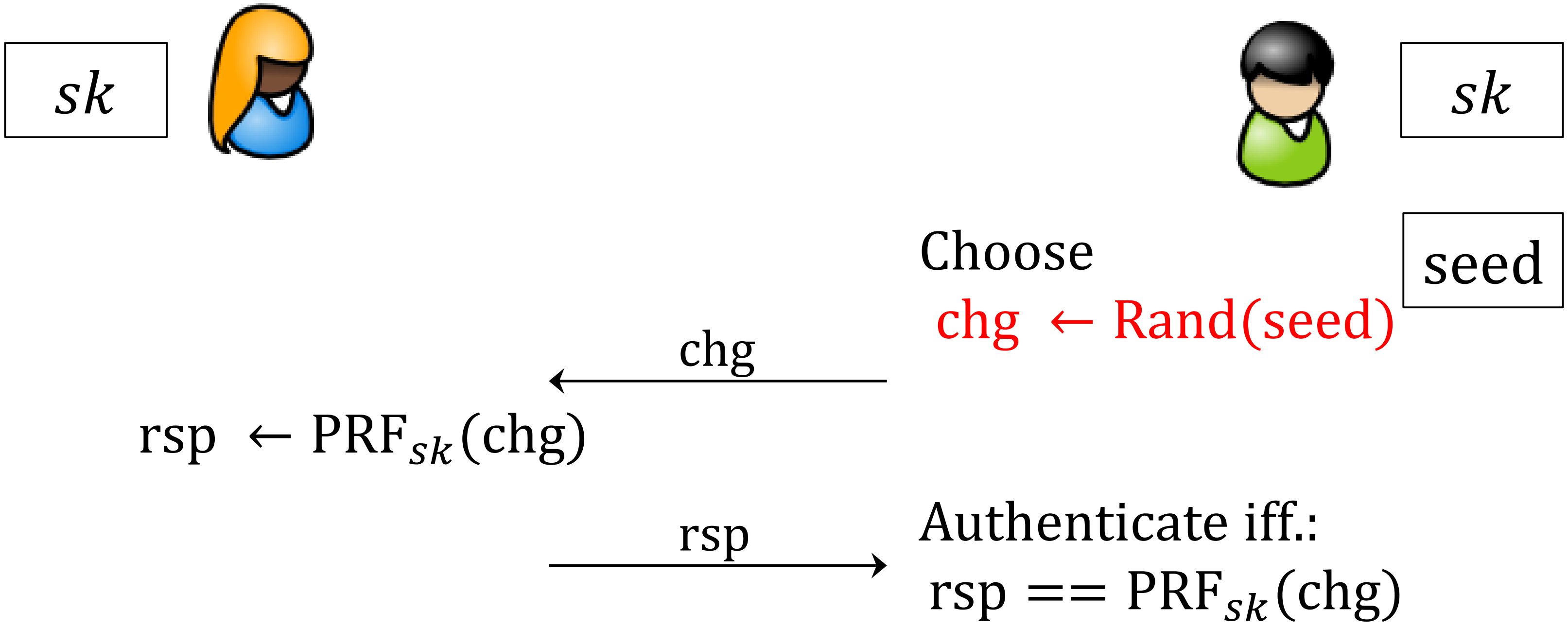
- Game $\mathbb{G}_0$: initial Impersonation Security game
- Game $\mathbb{G}_1$: Change game $\mathbb{G}_0$ to replace output of PRG in protocol by truly random numbers

- It holds that:

$$\mathbb{P}[A \text{ wins } \mathbb{G}_0] \leq \mathbb{P}[A \text{ wins } G_1] + (\mathbb{P}[A^* \text{dist.}\, \mathbb{G}_0 \,\&\, \mathbb{G}_1] - \frac{1}{2})$$

- What does it mean, distinguishing between $\mathbb{G}_0, \mathbb{G}_1$?
  - B plays the ImpSec game with the true output of PRF if $b = 1$ and with random output otherwise
  - At the end B has to guess the value of b

An Authentication Protocol



$\boxed{sk}$

$\boxed{sk}$

$\boxed{\text{seed}}$

Choose
$\text{chg} \leftarrow \text{Rand}(\text{seed})$

$\xleftarrow{\text{chg}}$

$\text{rsp} \leftarrow \text{PRF}_{sk}(\text{chg})$

$\xrightarrow{\text{rsp}}$

Authenticate iff.:
$\text{rsp} == \text{PRF}_{sk}(\text{chg})$

- The seed is only generated once
- Take PRG: $\{0,1\}^{|\text{seed}|} \to \{0,1\}^n$ PRF: $\{0,1\}^{|sk|} \times \{0,1\}^n \to \{0,1\}^m$

GAME HOP #1

- Game $\mathbb{G}_0$: initial Impersonation Security game
- Game $\mathbb{G}_1$: Change game $\mathbb{G}_0$ to replace output of PRG by truly random numbers

- Statement:

$$\mathbb{P}[A \text{ wins } \mathbb{G}_0] \leq \mathbb{P}[A \text{ wins } G_1] + (\mathbb{P}[A^* \text{dist.} \mathbb{G}_0 \,\&\, \mathbb{G}_1] - \frac{1}{2})$$

- Why is this true?
  - In fact, the only limitation A has in game $\mathbb{G}_1$ is the fact that it takes as input truly-random numbers; this translates to the advantage of adversary A*

Game hop #1

- Game $\mathbb{G}_0$: initial Impersonation Security game
- Game $\mathbb{G}_1$: Change game $\mathbb{G}_0$ to replace output of PRG by truly random numbers

- Statement:

$$\mathbb{P}[A \text{ wins } \mathbb{G}_0] \leq \mathbb{P}[A \text{ wins } G_1] + (\mathbb{P}[A^* \text{dist.} \, \mathbb{G}_0 \,\&\, \mathbb{G}_1] - \frac{1}{2})$$

- Statement:
  - $\mathbb{P}[A^* \text{dist.} \, \mathbb{G}_0 \,\&\, \mathbb{G}_1] = \frac{1}{2} + \epsilon_{\text{PRG}}$

GAME HOP #1

- Game $\mathbb{G}_0$: initial Impersonation Security game
- Game $\mathbb{G}_1$: Change game $\mathbb{G}_0$ to replace output of PRG by truly random numbers

- Statement:

$$\mathbb{P}[A \text{ wins } \mathbb{G}_0] \leq \mathbb{P}[A \text{ wins } G_1] + \epsilon_{\text{PRG}}$$

- Proof:
  - Suppose there exists A* distinguishing $\mathbb{G}_0, \mathbb{G}_1$.
  - Construct B distinguishing output of PRG from random

GAME HOP #1

- Proof:
  - Suppose there exists A* distinguishing $\mathbb{G}_0, \mathbb{G}_1$.
  - Construct B distinguishing output of PRG from random

  - B has to simulate the game for A*
  - The challenger for B generates *seed*
  - B generates *sk* & can make PRG queries to its challenger
  - NewSession queries: B returns an identifier $\underline{\pi}$
  - Send(V, prompt) : B makes PRG query, returns output to A*
  - Send(P, chg) : B computes PRF on A* 's input
  - Send(V, rsp) : B returns $\underline{\perp}$ to A*; it computes $\underline{1}$ iff. for that session rsp = PRF_{sk}(chg); else it computes 0
  - Result($*$): Return authentication bit if computed, or $\underline{\perp}$

Proof of Game Hop #1

- Proof:
  - The challenger for B generates seed and random bit b
  - B generates sk & can make PRG queries to its challenger
  - NewSession queries: B returns an identifier $\underline{\pi}$
  - Send(V,prompt) : B makes PRG query, returns output to A*
  - Send(P,chg) : B computes PRF on A* 's input
  - Send(V,rsp) : B returns $\underline{\perp}$ to A*; it computes $\underline{1}$ iff. for that session rsp = PRF_{sk}(chg); else it computes 0
  - Result($*$): Return authentication bit if computed, or $\underline{\perp}$
  - Finally, A* returns a guess d^* (0 if A* thinks it's playing game $\mathbb{G}_0$, 1 otherwise)
  - B receives this bit and sends its challenger a guess $\overline{d^*}$

Proof of Game Hop #1

- Analysis:
  - The challenger for B generates *seed* and random bit b
  - B generates sk & can make PRG queries to its challenger
  - NewSession queries: B returns an identifier $\underline{\pi}$
  - Send(V, prompt) : B makes PRG query, returns output to A*
  - Send(P, chg) : B computes PRF on A* 's input
  - Send(V, rsp) : B returns $\underline{\perp}$ to A*; it computes $\underline{1}$ iff. for that session rsp $=$ PRF_{sk}(chg); else it computes 0
  - Result($\ast$): Return authentication bit if computed, or $\underline{\perp}$
  - Finally, A* returns a guess d^* (0 if A* thinks it's playing game $\mathbb{G}_0$, 1 otherwise)
  - B receives this bit and sends its challenger a guess $\overline{d^*}$
  - If B's challenger chooses $b = 1$, then it outputs PRG queries, and B simulates perfectly $\mathbb{G}_0$; else, it perfectly simulates game $\mathbb{G}_1$
  - B succeeds as well as A succeeds

Game Hop #1

- Conclusion:

$$\mathbb{P}\left[A^{*}\text{ distinguishes }{}^{\mathbb{G}_0}/_{\mathbb{G}_1}\right] = \mathbb{P}[B\text{ wins PRG game}]$$

$$= \frac{1}{2} + \epsilon_{\mathrm{PRG}}$$

- Which leads to:

$$\mathbb{P}[A\text{ wins }\mathbb{G}_0] \leq \mathbb{P}[A\text{ wins }G_1] + \epsilon_{\mathrm{PRG}}$$

Game hop #2

- Game $\mathbb{G}_0$: initial Impersonation Security game
- Game $\mathbb{G}_1$: Change game $\mathbb{G}_0$ to replace output of PRG by truly random numbers
- Game $\mathbb{G}_2$: Change game $\mathbb{G}_1$ such that the truly random numbers output in $\mathbb{G}_1$ never repeat

- We had:

$$\mathbb{P}[A \text{ wins } \mathbb{G}_0] \leq \mathbb{P}[A \text{ wins } G_1] + \epsilon_{\text{PRG}}$$

- Statement:

$$\mathbb{P}[A \text{ wins } \mathbb{G}_1] \leq \mathbb{P}[A \text{ wins } G_2] + \binom{q_V}{2} 2^{-|\text{chg}|}$$

PROOF GAME HOP #2

- Game $\mathbb{G}_0$: initial Impersonation Security game
- Game $\mathbb{G}_1$: Change game $\mathbb{G}_0$ to replace output of PRG
  by truly random numbers
- Game $\mathbb{G}_2$: Change game $\mathbb{G}_1$ such that the truly random numbers output in $\mathbb{G}_1$ never repeat

- Statement:

$$\mathbb{P}[A \text{ wins } \mathbb{G}_1] \leq \mathbb{P}[A \text{ wins } G_2] + \binom{q_V}{2} 2^{-|\mathrm{chg}|}$$

- Proof:
  - Games $\mathbb{G}_1$ and $\mathbb{G}_2$ are indistinguishable unless there is a collision of randomness in 2 out of a total of q_V sessions

GAME HOP #3

- Game $\mathbb{G}_0$: initial Impersonation Security game
- Game $\mathbb{G}_1$: Replace PRG by truly random generator
- Game $\mathbb{G}_2$: Eliminate collision in PRG output
- Game $\mathbb{G}_3$: Same as game $\mathbb{G}_2$ except we abort if A anticipates PRG output before actually seeing it

  That is, A guesses a challenge in advance
- Statement:

$$\mathbb{P}[A \text{ wins } \mathbb{G}_2] \leq \mathbb{P}[A \text{ wins } G_3] + q_P \cdot 2^{-|\text{chg}|}$$

- Proof:
  - Games $\mathbb{G}_2$ and $\mathbb{G}_3$ are indistinguishable unless we abort. That happens w.p. $2^{-|\text{chg}|}$ for each of the q_P sessions

Game hop #4

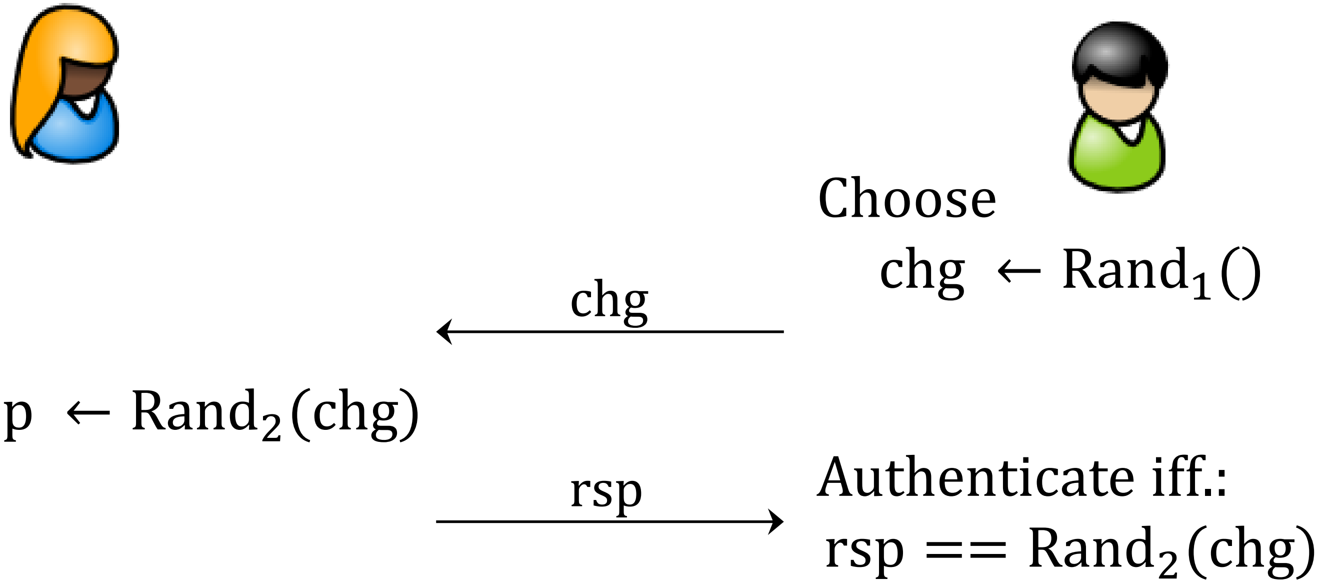
- Game $\mathbb{G}_0$: initial Impersonation Security game
- Game $\mathbb{G}_1$: Replace PRG by truly random generator
- Game $\mathbb{G}_2$: Eliminate collision in PRG output
- Game $\mathbb{G}_3$: Eliminate chg-guessing probability
- Game $\mathbb{G}_4$: Replace PRF output by truly random values

- Statement:

$$\mathbb{P}[A \text{ wins } \mathbb{G}_3] \leq \mathbb{P}[A \text{ wins } G_4] + \epsilon_{\text{PRF}}$$

- Proof:
  - Games $\mathbb{G}_3$ and $\mathbb{G}_4$ are indistinguishable except if there exists an adversary to distinguish the output of PRF from random – like the reduction $\mathbb{G}_0$ to $\mathbb{G}_1$

Where we are



Choose

chg $\leftarrow$ $\text{Rand}_1()$

chg

rsp $\leftarrow$ $\text{Rand}_2(\text{chg})$

rsp

Authenticate iff.:

rsp == $\text{Rand}_2(\text{chg})$

- At every step we idealized the protocol little by little
- Game $\mathbb{G}_4$: Take each of A's q_V sessions with the verifier

  A can only win if it sends rsp for fresh chg

  However: rsp is truly random; so A can only guess

Concluding the proof

- We had:

$$\mathbb{P}[A \text{ wins } \mathbb{G}_0] \leq \mathbb{P}[A \text{ wins } G_1] + \epsilon_{\text{PRG}}$$

$$\mathbb{P}[A \text{ wins } \mathbb{G}_1] \leq \mathbb{P}[A \text{ wins } G_2] + \binom{q_V}{2} 2^{-|\text{chg}|}$$

$$\mathbb{P}[A \text{ wins } \mathbb{G}_2] \leq \mathbb{P}[A \text{ wins } G_3] + q_P \cdot 2^{-|\text{chg}|}$$

$$\mathbb{P}[A \text{ wins } \mathbb{G}_3] \leq \mathbb{P}[A \text{ wins } G_4] + \epsilon_{\text{PRF}}$$

$$\mathbb{P}[A \text{ wins } \mathbb{G}_4] = q_V \cdot 2^{-|\text{rsp}|}$$

- Putting it all together:

$$\mathbb{P}[A \text{ wins } \mathbb{G}_0] \leq \epsilon_{\text{PRG}} + \binom{q_V}{2} 2^{-|\text{chg}|} + \epsilon_{\text{PRF}} + q_P(2^{-|\text{chg}|} + 2^{-|\text{rsp}|})$$

Part III
The Simulation Approach

GAME-BASED MODELS

- Adversary plays the game against the challenger
- Learning is done by Oracle access to primitives
- There is a challenge request from the adversary, a challenge from C, and a response from A
- Winning: the response "fits" the challenge and we rule out some attacks (response or learning have to have certain forms/bounds)
- Trivial attacks: basically, guessing the correct input for the response and/or challenge

Simulation-based models

- Usually express a non-discrete security guarantee :
  - "The adversary can't learn *anything* about a plaintext except its length"
  - "The adversary learns nothing about the identity of the authenticating user"

- Usually the attack is designed to protect data from one or more of the participants
  - Insider attacks: the adversary is one of the parties
  - Collusion attacks: one or multiple attackers collude with each other – for instance one insider and many outsiders
- Much harder to prove security in such a model

TYPICAL EXAMPLES

- Secure multi-party computation
  - Goal: two or more parties compute some output based on input from all parties
  - This is not always the same output
  - None of the parties learns anything about the other parties' input

- Typical example: the 2 millionaires' problem:
  - Two rich people want to know which one of them is richer, but without revealing their fortunes
  - Output to compute: the expression "a > b" or "b > a"
  - Sometimes (in the case of some wedding rituals) offering too little dowry or giving the father too little by way of a present may result in homicide!

Typical Security Definition

- Two types of adversaries:
  - Honest-but-curious: this adversary does not deviate from protocol, but will try to learn what it can from the output
  - Malicious: such adversaries may even deviate from protocol (for instance by choosing input different from their own, or deviant from honest distributions) in order to learn something about the other parties' inputs

2-PARTY HBC DEFINITIONS

- Ingredients
  - Functionality: $f: \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$
  - It takes 2 inputs (one from each party)
  - And outputs 2 outputs (one for each party)

- View:
  - Each party has a view, consisting of their secret, denoted w_i, their randomness, denoted r^i and the messages they receive $m^{i,1}, \ldots, m^{i,n}$ for each protocol execution
- Output:
  - Given its view, each participant computes Output^i

2-PARTY HBC DEFINITION

- A protocol π securely computes functionality $f = (f_1, f_2)$ in the presence of static semi-honest adversaries if there exist simulators S_1, S_2 such that:

$$\{(S_1\left(1^\lambda, x, f_1(x,y)\right), f(x,y)\}_{x,y,\lambda} \cong_C \{(view_1(x,y,n), \text{output}(x,y,n)\}_{x,y,\lambda}$$

$$\{(S_2\left(1^\lambda, y, f_2(x,y)\right), f(x,y)\}_{x,y,\lambda} \cong_C \{(view_2(x,y,n), \text{output}(x,y,n)\}_{x,y,\lambda}$$

for any x,y of same size