Applied Cryptography
Week 2: Randomness and Cryptographic Security

Bernardo Portela

M:ERSI, M:SI - 23

Context

- Last week we used and generated keys
- How is this done?

Context

- Last week we used and generated keys
- How is this done?

For Symmetric Crypto

- Generated uniformly at random
- Derived using a Key Derivation Function
  - From a password or low entropy secret
  - From a high-entropy master key from key exchange protocol

Context

- Last week we used and generated keys
- How is this done?

For Symmetric Crypto

- Generated uniformly at random
- Derived using a Key Derivation Function
  - From a password or low entropy secret
  - From a high-entropy master key from key exchange protocol

For Asymmetric Crypto

- Key generation algorithm $\rightarrow$ key pair
- Private key holder generates both keys; publishes public key
- Asymmetric keys are typically much larger
  - RSA keys take roughly 4096-bits for 128-bit security
  - Elliptic-curve keys take roughly 400-bits for 128-bit security

Storage and Generation

Keys are often *the most sensitive material* a secure system holds

Storage and Generation

Keys are often *the most sensitive material* a secure system holds

Ideally, in an external secure hardware

- Hardware Security Module (HSM)
- Smartcard or similar cryptographic token

Storage and Generation

Keys are often *the most sensitive material* a secure system holds

Ideally, in an external secure hardware

- Hardware Security Module (HSM)
- Smartcard or similar cryptographic token

Key wrapping

- Long-term keys are often *wrapped* before storage
- To encrypt with another key
- Password-based encryption (low security)
- Wrap with HW-protected master key (standard security)
- Master key stored in trusted hardware (high security)

To Be Random

Q1: Which of these numbers are random?

1. 00000000
2. 10101010
3. 00100100
4. 10011101

To Be Random

Q1: Which of these numbers are random?

1. 00000000 - Not random!
2. 10101010 - Not random (pattern)
3. 00100100 - Maybe not random?
4. 10011101 - Seems random...

To Be Random

Q1: Which of these numbers are random?

1. 00000000 - Not random!
2. 10101010 - Not random (pattern)
3. 00100100 - Maybe not random?
4. 10011101 - Seems random...

Randomness is not a property of a bit string, but rather:

- The bit generation process
- The bit string sampling procedure

To Be Random

Q1: Which of these numbers are random?

1. 00000000 - Not random!
2. 10101010 - Not random (pattern)
3. 00100100 - Maybe not random?
4. 10011101 - Seems random...

Randomness is not a property of a bit string, but rather:

- The bit generation process
- The bit string sampling procedure

Q2: Which of these numbers will more likely appear in a fair randomness generator?

Randomness Distributions

Randomized processes described using *randomness distributions*.

We start with the **uniform distribution** over a finite field S.

A process U samples from the uniform distribution if

$$\forall s^* \in S, \Pr[s = s^* : s \leftarrow_{\$} U] = \frac{1}{|S|}$$

Randomness Distributions

Randomized processes described using *randomness distributions*.

We start with the **uniform distribution** over a finite field S.

A process U samples from the uniform distribution if

$$\forall s^* \in S, \Pr[s = s^* : s \leftarrow_{\$} U] = \frac{1}{|S|}$$

Q1: If we roll a fair dice, what is the probability of getting 1?

Randomness Distributions

Randomized processes described using *randomness distributions*.

We start with the **uniform distribution** over a finite field S.

A process U samples from the uniform distribution if

$$\forall s^* \in S, \Pr[s = s^* : s \xleftarrow{\$} U] = \frac{1}{|S|}$$

Q1: If we roll a fair dice, what is the probability of getting 1?

$$\frac{1}{6} \approx 0.1667$$

Randomness Distributions

Randomized processes described using *randomness distributions*.

We start with the **uniform distribution** over a finite field S.

A process U samples from the uniform distribution if

$$\forall s^* \in S, \Pr[s = s^* : s \leftarrow_\$ U] = \frac{1}{|S|}$$

Q1: If we roll a fair dice, what is the probability of getting 1?

$$\frac{1}{6} \approx 0.1667$$

Q2: If we do a fair sampling of a byte, what is the probability of getting 00000000 **or** 10011101?

Randomness Distributions

Randomized processes described using *randomness distributions*.

We start with the **uniform distribution** over a finite field S.

A process U samples from the uniform distribution if

$$\forall s^* \in S, \Pr[s = s^* : s \leftarrow_{\$} U] = \frac{1}{|S|}$$

Q1: If we roll a fair dice, what is the probability of getting 1?

$$\tfrac{1}{6} \approx 0.1667$$

Q2: If we do a fair sampling of a byte, what is the probability of getting 00000000 **or** 10011101?

$$\tfrac{2}{2^8} \approx 0.0078$$

Quantifying Randomness

As you might have inferred, for the uniform sampling of λ bits, the probability of each element in the set is $\frac{1}{2^\lambda}$.

Quantifying Randomness

As you might have inferred, for the uniform sampling of λ bits, the probability of each element in the set is $\frac{1}{2^\lambda}$.

We do not always want to generate "nicely structured" bit strings

- E.g. a value from $0 \ldots 254$
- How to use uniformly generated bytes for this?

Quantifying Randomness

As you might have inferred, for the uniform sampling of λ bits, the probability of each element in the set is $\frac{1}{2^\lambda}$.

We do not always want to generate "nicely structured" bit strings

- E.g. a value from $0 \ldots 254$
- How to use uniformly generated bytes for this?

Q1: Get a byte, compute the result $\bmod 255$**. Is it uniform?**

Quantifying Randomness

As you might have inferred, for the uniform sampling of λ bits, the probability of each element in the set is $\frac{1}{2^\lambda}$.

We do not always want to generate "nicely structured" bit strings

- E.g. a value from $0 \ldots 254$
- How to use uniformly generated bytes for this?

Q1: Get a byte, compute the result $\bmod 255$**. Is it uniform?**

Bad corner case: bytes 0 and 255 both give us 0!

Quantifying Randomness

As you might have inferred, for the uniform sampling of λ bits, the probability of each element in the set is $\frac{1}{2^\lambda}$.

We do not always want to generate "nicely structured" bit strings

- E.g. a value from $0 \ldots 254$
- How to use uniformly generated bytes for this?

Q1: Get a byte, compute the result $\bmod 255$**. Is it uniform?**

Bad corner case: bytes 0 and 255 both give us 0!

Q2: Get a byte, exclude value 255 **and retry. Is it uniform?**

Quantifying Randomness

As you might have inferred, for the uniform sampling of λ bits, the probability of each element in the set is $\frac{1}{2^\lambda}$.

We do not always want to generate "nicely structured" bit strings

- E.g. a value from $0 \ldots 254$
- How to use uniformly generated bytes for this?

Q1: Get a byte, compute the result $\bmod 255$**. Is it uniform?**

Bad corner case: bytes 0 and 255 both give us 0!

Q2: Get a byte, exclude value 255 **and retry. Is it uniform?**

It is, and is called *rejection sampling*. **Q3: what is the downside?**

Entropy

We will mostly use *entropy* as an intuitive concept

- It measures uncertainty w.r.t. a sampling output

Entropy

We will mostly use *entropy* as an intuitive concept

- It measures uncertainty w.r.t. a sampling output

Mathematically, it can be defined for a distribution X as

$$H(X) = \sum_{s^* \in S} -\Pr[s^*] \cdot \log_b(\Pr[s])$$

Entropy

We will mostly use *entropy* as an intuitive concept

- It measures uncertainty w.r.t. a sampling output

Mathematically, it can be defined for a distribution X as

$$H(X) = \sum_{s^* \in S} -\Pr[s^*] \cdot \log_b(\Pr[s])$$

- It is maximized by the uniform distribution, with entropy λ

$$2^8 \cdot (-\frac{1}{2^8} \cdot \log_2(\frac{1}{2^8})) = 8$$

- Entropy here quantifies the number of uncertainty bits
  - In this example, we are uncertain of exactly 8 bits
- If a sampling is biased, it has less uncertainty, i.e. entropy

Random Number Generators

How do we get uniform coins?

Random Number Generators

How do we get uniform coins?

- It starts with a physical process
  - A source of entropy, e.g., some natural process that is believed to sample l-bits from a high-entropy distribution
  - Typically $l >> \lambda$ where λ is the assumed entropy
  - Randomness extractors (often a hash function) compress such bit strings down to λ bits
  - The result bit strings are assumed to be uniform

Random Number Generators

How do we get uniform coins?

- It starts with a physical process
  - A source of entropy, e.g., some natural process that is believed to sample l-bits from a high-entropy distribution
  - Typically $l >> \lambda$ where λ is the assumed entropy
  - Randomness extractors (often a hash function) compress such bit strings down to λ bits
  - The result bit strings are assumed to be uniform
- The combined process is called a Random Number Generator
- High-security RNGs currently exploit quantum effects

Pseudorandom Generators - Part 1

Good randomness is hard to generate, so RNGs are usually slow

Pseudorandom Generators are crypto's response to this problem:

- PRG takes a small, uniform seed of length λ
- Generates long, random-looking bit strings $l >> \lambda$
- PRGs are deterministic algorithms!

Pseudorandom Generators - Part 1

Good randomness is hard to generate, so RNGs are usually slow

Pseudorandom Generators are crypto's response to this problem:

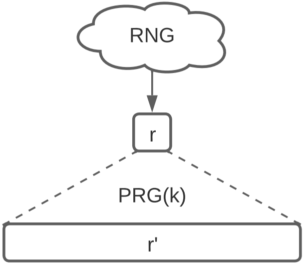
- PRG takes a small, uniform seed of length λ
- Generates long, random-looking bit strings $l >> \lambda$
- PRGs are deterministic algorithms!

A Pseudorandom generator is a function $G : \{0,1\}^{\lambda} \rightarrow \{0,1\}^{l}$

Security: (without delving deep in probability) an attacker must be unable of distinguishing PRG outputs from a truly random string

Pseudorandom Generators - Part 2

$$PRG : \{0,1\}^{\lambda} \rightarrow \{0,1\}^{l}$$

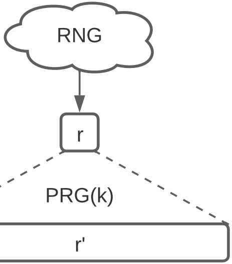


Reasoning

- Use a strong RNG to generate seed r of (small) size λ
- Use the PRG on seed r to generate (much larger) r' of size l

Pseudorandom Generators - Part 2

$$PRG : \{0,1\}^\lambda \rightarrow \{0,1\}^l$$



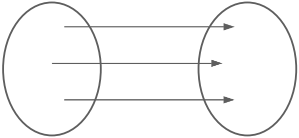
Reasoning

- Use a strong RNG to generate seed r of (small) size λ
- Use the PRG on seed r to generate (much larger) r' of size l

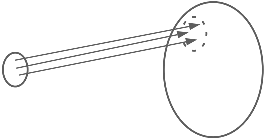
Q: Can we have secure PRGs (indistinguishable from uniform distribution), considering adversaries with unbound power?

Security of Pseudorandom Generators

$$U : \{0,1\}^l \rightarrow \{0,1\}^l$$



$$PRG : \{0,1\}^\lambda \rightarrow \{0,1\}^l$$



- An adversary can simply test all 2^λ cases
- Security refers to a computationally limited adversary
- One that cannot (realistically) test all possible PRG inputs

Security in Practice

Redefine "impossible to break"

- With *reasonable resources* (time, memory, HW power)
- With probability higher than *negligible*

Security in Practice

Redefine "impossible to break"

- With *reasonable resources* (time, memory, HW power)
- With probability higher than *negligible*

Practical schemes are *computationally impossible* to break

Take an encryption scheme and an attacker that does not know k

- Attacker chooses non-repeating inputs X_i and gets
  - Y_i chosen uniformly at random if $b = 1$
  - $Y_i = E(k, X_i)$ if $b = 0$
- Attacker guesses b and wins if $b = b'$

Security in Practice

Redefine "impossible to break"

- With *reasonable resources* (time, memory, HW power)
- With probability higher than *negligible*

Practical schemes are *computationally impossible* to break

Take an encryption scheme and an attacker that does not know k

- Attacker chooses non-repeating inputs X_i and gets
  - Y_i chosen uniformly at random if $b = 1$
  - $Y_i = E(k, X_i)$ if $b = 0$
- Attacker guesses b and wins if $b = b'$

We define the adversary's advantage ϵ as

$$\epsilon = |\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]|$$

Best attack for $\epsilon = 2^{-40}$ takes 2^{80} steps

Concrete Numbers - Part 1

Some numbers for scale

- Not easy to perceive very very large numbers
- The estimated age of the universe in nanosecs is around 2^{88}
- The number of atoms in the universe is roughly 2^{256}

Concrete Numbers - Part 1

Some numbers for scale

- Not easy to perceive very very large numbers
- The estimated age of the universe in nanosecs is around 2^{88}
- The number of atoms in the universe is roughly 2^{256}

A common security parameter

- A common size for keys is 128 bits
- Consider the following events
  - Winning a lottery with 9 million participants (all of Portugal)
  - Guessing a 2^{128} size key at the first try

Concrete Numbers - Part 1

Some numbers for scale

- Not easy to perceive very very large numbers
- The estimated age of the universe in nanosecs is around 2^{88}
- The number of atoms in the universe is roughly 2^{256}

A common security parameter

- A common size for keys is 128 bits
- Consider the following events
  - Winning a lottery with 9 million participants (all of Portugal)
  - Guessing a 2^{128} size key at the first try

Q1: Which event is more likely?

Concrete Numbers - Part 1

Some numbers for scale

- Not easy to perceive very very large numbers
- The estimated age of the universe in nanosecs is around 2^{88}
- The number of atoms in the universe is roughly 2^{256}

A common security parameter

- A common size for keys is 128 bits
- Consider the following events
  - Winning a lottery with 9 million participants (all of Portugal)
  - Guessing a 2^{128} size key at the first try

Q1: Which event is more likely?

Q2: By how much?

Concrete Numbers - Part 2

Security is defined as (t, ϵ)-security

- For some well-defined attack model
- Any attacker must run in at most t steps
- Has at most ϵ success advantage/probability
- t is a lower-bound on the work needed to break the scheme

Concrete Numbers - Part 2

Security is defined as (t, ϵ)-security

- For some well-defined attack model
- Any attacker must run in at most t steps
- Has at most ϵ success advantage/probability
- t is a lower-bound on the work needed to break the scheme

Define security of the best possible encryption with key space 2^{128}

Q1: For $t = 2^{128}$, what is ϵ?

Concrete Numbers - Part 2

Security is defined as (t, ϵ)-security

- For some well-defined attack model
- Any attacker must run in at most t steps
- Has at most ϵ success advantage/probability
- t is a lower-bound on the work needed to break the scheme

Define security of the best possible encryption with key space 2^{128}

Q1: For $t = 2^{128}$, what is ϵ? $\epsilon = 1$

Q2: For $t = 1$, what is ϵ?

Concrete Numbers - Part 2

Security is defined as (t, ϵ)-security

- For some well-defined attack model
- Any attacker must run in at most t steps
- Has at most ϵ success advantage/probability
- t is a lower-bound on the work needed to break the scheme

Define security of the best possible encryption with key space 2^{128}

Q1: For $t = 2^{128}$, what is ϵ? $\epsilon = 1$

Q2: For $t = 1$, what is ϵ? $\epsilon = 2^{-128}$

Q3: For $t = 2^{64}$, what is ϵ?

Concrete Numbers - Part 2

Security is defined as (t, ϵ)-security

- For some well-defined attack model
- Any attacker must run in at most t steps
- Has at most ϵ success advantage/probability
- t is a lower-bound on the work needed to break the scheme

Define security of the best possible encryption with key space 2^{128}

Q1: For $t = 2^{128}$, what is ϵ? $\epsilon = 1$

Q2: For $t = 1$, what is ϵ? $\epsilon = 2^{-128}$

Q3: For $t = 2^{64}$, what is ϵ? $\epsilon = 2^{-64}$

The more tries you get, the greater ϵ becomes: $(t, t/2^{128})$ security

Quantifying Security

Lower bound on the work required for a successful attack

Number of steps of the best attack

- n-bits security
- Best attack to break the scheme requires 2^n steps
- n-bit keys cannot ever give more than n-bit security

Quantifying Security

Lower bound on the work required for a successful attack

Number of steps of the best attack

- n-bits security
- Best attack to break the scheme requires 2^n steps
- n-bit keys cannot ever give more than n-bit security
  - **Q1: Why?**

Quantifying Security

Lower bound on the work required for a successful attack

Number of steps of the best attack

- n-bits security
- Best attack to break the scheme requires 2^n steps
- n-bit keys cannot ever give more than n-bit security
  - **Q1: Why?**
- Brute-force attack allows finding the correct key
- l-bit keys could lead to n-bit security s.t. $n << t$

Quantifying Security

Lower bound on the work required for a successful attack

Number of steps of the best attack

- *n*-bits security
- Best attack to break the scheme requires 2^n steps
- *n*-bit keys cannot ever give more than *n*-bit security
  - **Q1: Why?**
- Brute-force attack allows finding the correct key
- *l*-bit keys could lead to *n*-bit security s.t. $n << t$
  - **Q2: When?**

Quantifying Security

Lower bound on the work required for a successful attack

Number of steps of the best attack

- *n*-bits security
- Best attack to break the scheme requires 2^n steps
- *n*-bit keys cannot ever give more than *n*-bit security
  - **Q1: Why?**
- Brute-force attack allows finding the correct key
- *l*-bit keys could lead to *n*-bit security s.t. $n << t$
  - **Q2: When?**
  - Best attack is more efficient than brute-force
  - Common in asymmetric cryptography
  - Keys must follow specific structures, not random bit strings
- Quantifying using *n*-bit security permits comparing schemes

Good Security Values for Real-world Crypto

The 2^{128} rule of thumb

- Designs for which best attacks are at $(t, \epsilon) = (2^{88}, 2^{-40})$

Good Security Values for Real-world Crypto

The 2^{128} rule of thumb

- Designs for which best attacks are at $(t, \epsilon) = (2^{88}, 2^{-40})$

For how long do we need security to hold?

- Moore's law: computational power doubles every 2 years
- $n + 1$ bit security every 2 years
- This no longer seems to be true, but...
- Maybe we will have quantum computers soon

Long-term security: $\approx$ 256-bit keys

Short-term security: $\approx$ 80-bit keys may be OK

Stateful PRGs in Operating Systems

Randomness generation is statful

- ... in modern OSs
- PRG keeps a state
- OS mixes output of entropy source into PRG state

Stateful PRGs in Operating Systems

Randomness generation is statful

- ... in modern OSs
- PRG keeps a state
- OS mixes output of entropy source into PRG state

Extract and expand randomness

- $st \leftarrow \text{init}()$: SO initializes state
- $st \leftarrow \text{refresh}(R, st)$: SO adds entropy (reseeds)
- $(C, st) \leftarrow \text{next}(N, st)$: SO returns N random bits

Dealing With a Compromised State

Backtracking $\Leftarrow$ resistance

- Suppose an adversary corrupts the PRG state
- Past randomness should not be compromised
  - We might have used it to generate cryptographic material
- A.k.a. *forward secrecy* (for past secret keys)

Dealing With a Compromised State

Backtracking $\Leftarrow$ resistance

- Suppose an adversary corrupts the PRG state
- Past randomness should not be compromised
  - We might have used it to generate cryptographic material
- A.k.a. *forward secrecy* (for past secret keys)

Prediction $\Rightarrow$ resistance

- Suppose the adversary corrupts the PRG state
- SO adds extra (hidden) entropy to PRG state
- Future output should look random once more
- Hence refresh must be called regularly

Linux systems

- PRG is accessible at */dev/urandom*
  - In ∗nix-style, PRG is mapped to a file
  - Careful to make sure system calls are successful!

Linux systems

- PRG is accessible at */dev/urandom*
  - In ∗nix-style, PRG is mapped to a file
  - Careful to make sure system calls are successful!

Link to code from **LibreSSL**

In some variants, there is a blocking */dev/random* based on an entropy simulator

- Check if there is “sufficient entropy”
- Blocks otherwise
- Current consensus indicates that, for most applications, this is not useful (see **this link** for more information)

Caution: statistical tests are not sufficient

- **Q: What type of tests can we do over "random" inputs?**

Caution: statistical tests are not sufficient

- **Q: What type of tests can we do over "random" inputs?**
  - Count number of 1s and 0s
  - Check distribution of 8-bit words
  - Look for patterns
  - ...

Irrelevant for Security

- Possible to pass statistical tests
- Totally insecure for cryptographic purposes

Caution: statistical tests are not sufficient

- **Q: What type of tests can we do over "random" inputs?**
  - Count number of 1s and 0s
  - Check distribution of 8-bit words
  - Look for patterns
  - ...

Irrelevant for Security

- Possible to pass statistical tests
- Totally insecure for cryptographic purposes

Cryptographic PRGs come with a *proof of security*

- Goal: Given n bits of input, can an adversary guess bit $n + 1$?
- Secure PRGs used directly, or as building blocks to other PRGs

Security Assurance

There are two main ways in which security is ensured:

- Heuristically
- Provably (not probably!)

Security Assurance

There are two main ways in which security is ensured:

- Heuristically
- Provably (not probably!)

Heuristic Security

- Large community constantly trying to break schemes
- Cryptanalysts trying to disprove n-bit security
- The AES block cipher is an example

Security Assurance

There are two main ways in which security is ensured:

- Heuristically
- Provably (not probably!)

Heuristic Security

- Large community constantly trying to break schemes
- Cryptanalysts trying to disprove n-bit security
- The AES block cipher is an example

Provable Security

- Mathematical proof
- Breaking a scheme implies solving a hard problem
- A mathematical problem, or breaking another scheme!

Provable Security

Assumption: mathematical problem P cannot be efficiently solved

Goal: Breaking scheme C cannot be efficiently done

Provable Security

Assumption: mathematical problem P cannot be efficiently solved

Goal: Breaking scheme C cannot be efficiently done

Methodology: building a reduction

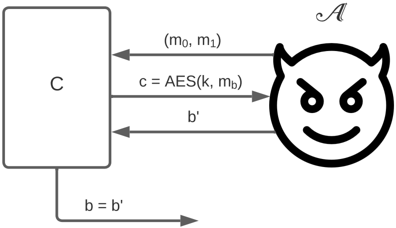
- Take any (hypothetical) attacker $\mathcal{A}$ that breaks C
- Construct (concrete) reduction $\mathcal{B}^{\mathcal{A}}$
- I.e. $\mathcal{B}$ uses $\mathcal{A}$ as a subroutine
- Show that $\mathcal{B}$ solves P when $\mathcal{A}$ succeeds

We never state that C is secure by itself

We state that C is as secure as the hardness of P

An Example of Provable Security - Part 1

Assume that AES is a semantic secure scheme, i.e.

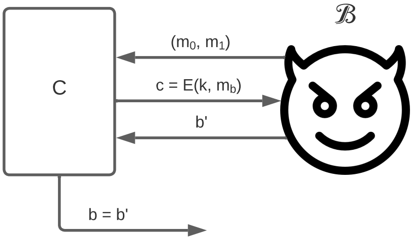


An adversary with non-negligible victory probability (over $\frac{1}{2}$), i.e a successful $\mathcal{A}$ must not exist!

An Example of Provable Security - Part 2

Consider an encryption scheme that just repeats AES 2 times.

$$E(k, m) = AES(k, m) \mid AES(k, m)$$

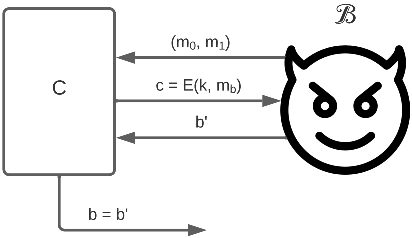


Q: given that AES is secure, is this secure?

An Example of Provable Security - Part 2

Consider an encryption scheme that just repeats AES 2 times.

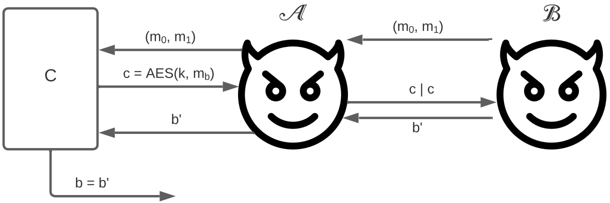
$$E(k, m) = AES(k, m) \mid AES(k, m)$$



Q: given that AES is secure, is this secure?

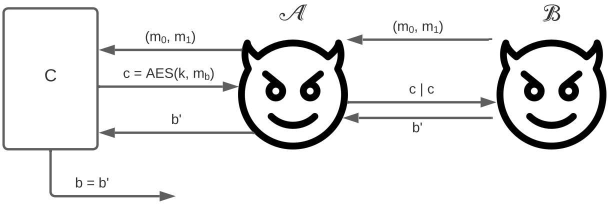
- It should be...
- We are just repeating the encryption
- Can we demonstrate this?

An Example of Provable Security - Part 3



- **Suppose a successful** $\mathcal{B}$ exists
- Then, we can construct a concrete $\mathcal{A}$ to break AES like this
- Contradiction! We assumed that no such $\mathcal{A}$ can exist!

An Example of Provable Security - Part 3



- **Suppose a successful** $\mathcal{B}$ exists
- Then, we can construct a concrete $\mathcal{A}$ to break AES like this
- Contradiction! We assumed that no such $\mathcal{A}$ can exist!

Corollary

- No $\mathcal{B}^{\mathcal{A}}$ can exist (AES is secure)
- As such, no $\mathcal{A}$ can exist
- So, scheme E must be secure!

Caveats of Provable Security

Problem P is called a *hardness assumption*

- It can be a mathematical problem, such as factoring
- It can be some other cryptogaphic construction

Caveats of Provable Security

Problem P is called a *hardness assumption*

- It can be a mathematical problem, such as factoring
- It can be some other cryptogaphic construction

Proof assurance $\leq$ assumption assurance

- Proofs of security are relative to assumptions
- Security only holds if assumptions are true

Most of the assumptions are validated via **heuristic security**

Heuristic Security

Validating hardness assumptions is crucial for modern cryptography

Methodology for heuristic security has been progressing

- Standards take years to define
- Competitions where proposals are scrutinized
  - It is how AES was established as the *de facto* encryption standard for the overwhelming majority of applications
  - And is how PQ encryption schemes are being selected
- "My construction wins if I break your construction"
  - Yet again we see the value of the Kerckhoffs's principle!

Applied Cryptography
Week 2: Randomness and Cryptographic Security

Bernardo Portela

M:ERSI, M:SI - 23