Random Oracles

- PRFs look random if you don't know the key.

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.
- PRGs look random if you don't know the input.

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.
- PRGs look random if you don't know the input.
  - Even though there is no key. (So anyone, even the adversary, can always evaluate it.)

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.
- PRGs look random if you don't know the input.
  - Even though there is no key. (So anyone, even the adversary, can always evaluate it.)
- With hash functions, both the key and the input are known.

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.
- PRGs look random if you don't know the input.
  - Even though there is no key. (So anyone, even the adversary, can always evaluate it.)
- With hash functions, both the key and the input are known.
  - What could it mean to say that this looks random?

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.
- PRGs look random if you don't know the input.
  - Even though there is no key. (So anyone, even the adversary, can always evaluate it.)
- With hash functions, both the key and the input are known.
  - What could it mean to say that this looks random?
  - Nothing! We don't claim the output looks random.

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.
- PRGs look random if you don't know the input.
  - Even though there is no key. (So anyone, even the adversary, can always evaluate it.)
- With hash functions, both the key and the input are known.
  - What could it mean to say that this looks random?
  - Nothing! We don't claim the output looks random.
  - We claim collisions are hard to find.

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.
- PRGs look random if you don't know the input.
  - Even though there is no key. (So anyone, even the adversary, can always evaluate it.)
- With hash functions, both the key and the input are known.
  - What could it mean to say that this looks random?
  - Nothing! We don't claim the output looks random.
  - We claim collisions are hard to find.

Random Oracle Model Design an algorithm or a protocol as though you have access to a random function.

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.
- PRGs look random if you don't know the input.
  - Even though there is no key. (So anyone, even the adversary, can always evaluate it.)
- With hash functions, both the key and the input are known.
  - What could it mean to say that this looks random?
  - Nothing! We don't claim the output looks random.
  - We claim collisions are hard to find.

Random Oracle Model Design an algorithm or a protocol as though you have access to a random function.
Assume the adversary has access to the same random function.

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.
- PRGs look random if you don't know the input.
  - Even though there is no key. (So anyone, even the adversary, can always evaluate it.)
- With hash functions, both the key and the input are known.
  - What could it mean to say that this looks random?
  - Nothing! We don't claim the output looks random.
  - We claim collisions are hard to find.

Random Oracle Model Design an algorithm or a protocol as though you have access to a random function.
Assume the adversary has access to the same random function.
When you prove security, assume there is an "oracle" that responds to all queries.

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.
- PRGs look random if you don't know the input.
  - Even though there is no key. (So anyone, even the adversary, can always evaluate it.)
- With hash functions, both the key and the input are known.
  - What could it mean to say that this looks random?
  - Nothing! We don't claim the output looks random.
  - We claim collisions are hard to find.

Random Oracle Model Design an algorithm or a protocol as though you have access to a random function.
Assume the adversary has access to the same random function.
When you prove security, assume there is an "oracle" that responds to all queries.
In the real world, just replace all queries with the output of a hash function.

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.
- PRGs look random if you don't know the input.
  - Even though there is no key. (So anyone, even the adversary, can always evaluate it.)
- With hash functions, both the key and the input are known.
  - What could it mean to say that this looks random?
  - Nothing! We don't claim the output looks random.
  - We claim collisions are hard to find.

Random Oracle Model Design an algorithm or a protocol as though you have access to a random function.
Assume the adversary has access to the same random function.
When you prove security, assume there is an "oracle" that responds to all queries.
In the real world, just replace all queries with the output of a hash function.
Hope for the best!

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.
- PRGs look random if you don't know the input.
  - Even though there is no key. (So anyone, even the adversary, can always evaluate it.)
- With hash functions, both the key and the input are known.
  - What could it mean to say that this looks random?
  - Nothing! We don't claim the output looks random.
  - We claim collisions are hard to find.

Random Oracle Model Design an algorithm or a protocol as though you have access to a random function.
Assume the adversary has access to the same random function.
When you prove security, assume there is an "oracle" that responds to all queries.
In the real world, just replace all queries with the output of a hash function.
Hope for the best!

1. Example: password hashing. $h = H(\text{pwd})$ reveals nothing, unless the adversary specifically evaluates H on input pwd.

Random Oracles

- PRFs look random if you don't know the key.
  - Even if you do know the input.
- PRGs look random if you don't know the input.
  - Even though there is no key. (So anyone, even the adversary, can always evaluate it.)
- With hash functions, both the key and the input are known.
  - What could it mean to say that this looks random?
  - Nothing! We don't claim the output looks random.
  - We claim collisions are hard to find.

Random Oracle Model Design an algorithm or a protocol as though you have access to a random function.
Assume the adversary has access to the same random function.
When you prove security, assume there is an "oracle" that responds to all queries.
In the real world, just replace all queries with the output of a hash function.
Hope for the best!

1. Example: password hashing. $h = H(\text{pwd})$ reveals nothing, unless the adversary specifically evaluates H on input pwd.
2. Example: key derivation. We need a uniform, random key. Sometimes we have non-uniform input, such as biometric data. $h = H(\text{iris})$ reveals nothing, unless the adversary specifically evaluates H on input iris.

Min Entropy

Definition

Let $\mathcal{D}$ be a distribution. The min-entropy of $\mathcal{D}$, measured in bits, is

$$H_\infty(\mathcal{D}) = -\log \max_x \Pr[x \leftarrow \mathcal{D}]$$

Min Entropy

Definition

Let $\mathcal{D}$ be a distribution. The min-entropy of $\mathcal{D}$, measured in bits, is

$$H_\infty(\mathcal{D}) = -\log \max_x \Pr[x \leftarrow \mathcal{D}]$$

Think of this as the *guessing probability.* To guess which item will be sampled, guess the item with the maximum probability under $\mathcal{D}$.

Min Entropy

Definition

Let $\mathcal{D}$ be a distribution. The min-entropy of $\mathcal{D}$, measured in bits, is

$$H_\infty(\mathcal{D}) = -\log \max_x \Pr[x \leftarrow \mathcal{D}]$$

Think of this as the *guessing probability.* To guess which item will be sampled, guess the item with the maximum probability under $\mathcal{D}$.
For example, if the most likely outcome has probability 2^{-n}, Then

$$H_\infty(\mathcal{D}) = -\log 2^{-n} = n.$$

Min Entropy

Definition

Let $\mathcal{D}$ be a distribution. The min-entropy of $\mathcal{D}$, measured in bits, is

$$H_\infty(\mathcal{D}) = -\log \max_x \Pr[x \leftarrow \mathcal{D}]$$

Password hashing:
Adversary sees $h = H(\text{pwd})$.
In 2009, the RockYou! pwd database, containing 32 million pwds, was breached.

Min Entropy

Definition

Let $\mathcal{D}$ be a distribution. The min-entropy of $\mathcal{D}$, measured in bits, is

$$H_\infty(\mathcal{D}) = -\log \max_x \Pr[x \leftarrow \mathcal{D}]$$

Password hashing:
Adversary sees $h = H(\text{pwd})$.
In 2009, the RockYou! pwd database, containing 32 million pwds, was breached.

Password	Count	Percentage
123456	290731	0.8918%
12345	79078	0.2426%
123456789	76790	0.2356%
password	59463	0.1824%
iloveyou	49952	0.1532%

Min Entropy

Definition

Let $\mathcal{D}$ be a distribution. The min-entropy of $\mathcal{D}$, measured in bits, is

$$H_\infty(\mathcal{D}) = -\log \max_x \Pr[x \leftarrow \mathcal{D}]$$

Password hashing:
Adversary sees $h = H(\text{pwd})$.
In 2009, the RockYou! pwd database, containing 32 million pwds, was breached.

Password	Count	Percentage
123456	290731	0.8918%
12345	79078	0.2426%
123456789	76790	0.2356%
password	59463	0.1824%
iloveyou	49952	0.1532%

Guess pwd $= 123456$. Ideally, $\Pr[\text{pwd} = 123456 \mid h] = .89\%$.
This would mean that the hash provided no additional information.

Min Entropy

Definition

Let $\mathcal{D}$ be a distribution. The min-entropy of $\mathcal{D}$, measured in bits, is

$$H_\infty(\mathcal{D}) = -\log \max_x \Pr[x \leftarrow \mathcal{D}]$$

Password hashing:
Adversary sees $h = H(\text{pwd})$.
In 2009, the RockYou! pwd database, containing 32 million pwds, was breached.

Password	Count	Percentage
123456	290731	0.8918%
12345	79078	0.2426%
123456789	76790	0.2356%
password	59463	0.1824%
iloveyou	49952	0.1532%

Guess pwd $= 123456$. Ideally, $\Pr[\text{pwd} = 123456 \mid h] = .89\%$.
This would mean that the hash provided no additional information.

Check if $h = H(123456)$. If not, then, ideally:
$\Pr[\text{pwd} = 12345 \mid \text{pwd} \neq 123456] = \frac{\Pr[\text{pwd}=12345 \wedge \text{pwd} \neq 123456]}{\Pr[\text{pwd} \neq 123456]} = \frac{\Pr[\text{pwd}=12345]}{\Pr[\text{pwd} \neq 123456]} = .27\%$.

Min Entropy

Definition

Let $\mathcal{D}$ be a distribution. The min-entropy of $\mathcal{D}$, measured in bits, is

$$H_\infty(\mathcal{D}) = -\log \max_x \Pr[x \leftarrow \mathcal{D}]$$

Password hashing:
Adversary sees $h = H(\text{pwd})$.
In 2009, the RockYou! pwd database, containing 32 million pwds, was breached.

Password	Count	Percentage
123456	290731	0.8918%
12345	79078	0.2426%
123456789	76790	0.2356%
password	59463	0.1824%
iloveyou	49952	0.1532%

Guess pwd $= 123456$. Ideally, $\Pr[\text{pwd} = 123456 \mid h] = .89\%$.
This would mean that the hash provided no additional information.

Check if $h = H(123456)$. If not, then, ideally:
$\Pr[\text{pwd} = 12345 \mid \text{pwd} \neq 123456] = \frac{\Pr[\text{pwd}=12345 \wedge \text{pwd} \neq 123456]}{\Pr[\text{pwd} \neq 123456]} = \frac{\Pr[\text{pwd}=12345]}{\Pr[\text{pwd} \neq 123456]} = .27\%.$
This would mean that nothing was learned, except that pwd $\neq 123456$.

PWDs using CRHF

Adversary sees $h = H(\text{pwd})$

Consider the following collision resistant hash function, $h^s : \{0,1\}^{2n} \rightarrow \{0,1\}^n$:

$\underline{h^s(b||x||y)}$:
If $b = 0 \wedge y = 0^n$, output $0||x$.
Else, output $1||\hat{h}^s(b||x||y)$, where $\hat{h}^s : \{0,1\}^{2n} \rightarrow \{0,1\}^{n-1}$ is a fixed-length CRHF.

PWDs using CRHF

Adversary sees $h = H(\text{pwd})$

Consider the following collision resistant hash function, $h^s : \{0,1\}^{2n} \rightarrow \{0,1\}^n$:

$h^s(b||x||y)$:
If $b = 0 \wedge y = 0^n$, output $0||x$.
Else, output $1||\hat{h}^s(b||x||y)$, where $\hat{h}^s : \{0,1\}^{2n} \rightarrow \{0,1\}^{n-1}$ is a fixed-length CRHF.

What is $h(\text{pwd})$, if pwd requires padding of 0^n?

PWDs using CRHF

Adversary sees $h = H(\text{pwd})$

Consider the following collision resistant hash function, $h^s : \{0,1\}^{2n} \rightarrow \{0,1\}^n$:

$\underline{h^s(b||x||y)}$:
If $b = 0 \wedge y = 0^n$, output $0||x$.
Else, output $1||\hat{h}^s(b||x||y)$, where $\hat{h}^s : \{0,1\}^{2n} \rightarrow \{0,1\}^{n-1}$ is a fixed-length CRHF.

What is $h(\text{pwd})$, if pwd requires padding of 0^n?

Consider the following collision resistant hash function, $h^s : \{0,1\}^{2n} \rightarrow \{0,1\}^n$:

$\underline{h^s(b||x||y)}$:
Output $b||\hat{h}^s(b||x||y)$, where $\hat{h}^s : \{0,1\}^{2n} \rightarrow \{0,1\}^{n-1}$ is a fixed-length CRHF.

What is $h(\text{pwd})$, if pwd requires padding of 0^n?

PWDs using CRHF

Adversary sees $h = H(\text{pwd})$

Consider the following collision resistant hash function, $h^s : \{0,1\}^{2n} \rightarrow \{0,1\}^n$:

$\underline{h^s(b||x||y)}$:
If $b = 0 \wedge y = 0^n$, output $0||x$.
Else, output $1||\hat{h}^s(b||x||y)$, where $\hat{h}^s : \{0,1\}^{2n} \rightarrow \{0,1\}^{n-1}$ is a fixed-length CRHF.

What is $h(\text{pwd})$, if pwd requires padding of 0^n?

Consider the following collision resistant hash function, $h^s : \{0,1\}^{2n} \rightarrow \{0,1\}^n$:

$\underline{h^s(b||x||y)}$:
Output $b||\hat{h}^s(b||x||y)$, where $\hat{h}^s : \{0,1\}^{2n} \rightarrow \{0,1\}^{n-1}$ is a fixed-length CRHF.

What is $h(\text{pwd})$, if pwd requires padding of 0^n?
Seeing $h(x)$ roughly doubles your probability of guessing pwd.

Random Oracle

Adversary sees $h = H(\text{pwd})$

Let's model H as a random oracle:

Random Oracle

Adversary sees $h = H(\text{pwd})$

Let's model H as a random oracle:
Each time $\mathcal{A}$ wants to evaluate H, it queries an "oracle" that holds a random function.

Random Oracle

Adversary sees $h = H(\text{pwd})$

Let's model H as a random oracle:
Each time $\mathcal{A}$ wants to evaluate H, it queries an "oracle" that holds a random function.
Because it is a random function, a wrong query reveals nothing about pwd.

Random Oracle

Adversary sees $h = H(\text{pwd})$

Let's model H as a random oracle:
Each time $\mathcal{A}$ wants to evaluate H, it queries an "oracle" that holds a random function.
Because it is a random function, a wrong query reveals nothing about pwd.
Since there are no random oracles in the sky, we then *instantiate* with some hash function.

Random Oracle

Adversary sees $h = H(\text{pwd})$

Let's model H as a random oracle:
Each time $\mathcal{A}$ wants to evaluate H, it queries an "oracle" that holds a random function.
Because it is a random function, a wrong query reveals nothing about pwd.
Since there are no random oracles in the sky, we then *instantiate* with some hash function.

Is this a reasonable assumption?

Random Oracle

Adversary sees $h = H(\text{pwd})$

Let's model H as a random oracle:
Each time $\mathcal{A}$ wants to evaluate H, it queries an "oracle" that holds a random function.
Because it is a random function, a wrong query reveals nothing about pwd.
Since there are no random oracles in the sky, we then *instantiate* with some hash function.

Is this a reasonable assumption?

- We know that there exist constructions of various cryptographic primitives (e.g. encryption schemes) that are secure in the RO model, but are provably *insecure, regardless of what hash function is used in the instantiation.*

Random Oracle

Adversary sees $h = H(\text{pwd})$

Let's model H as a random oracle:
Each time $\mathcal{A}$ wants to evaluate H, it queries an "oracle" that holds a random function.
Because it is a random function, a wrong query reveals nothing about pwd.
Since there are no random oracles in the sky, we then *instantiate* with some hash function.

Is this a reasonable assumption?

- We know that there exist constructions of various cryptographic primitives (e.g. encryption schemes) that are secure in the RO model, but are provably *insecure, regardless of what hash function is used in the instantiation.*
  - However, these constructions are contrived.

Random Oracle

Adversary sees $h = H(\text{pwd})$

Let's model H as a random oracle:
Each time $\mathcal{A}$ wants to evaluate H, it queries an "oracle" that holds a random function.
Because it is a random function, a wrong query reveals nothing about pwd.
Since there are no random oracles in the sky, we then *instantiate* with some hash function.

Is this a reasonable assumption?

- We know that there exist constructions of various cryptographic primitives (e.g. encryption schemes) that are secure in the RO model, but are provably *insecure, regardless of what hash function is used in the instantiation.*
  - However, these constructions are contrived.
- We have seen a MAC scheme that was secure in the RO, but insecure if the hash function instantiating the RO followed the Merkle-Damgard transform.

Random Oracle

Adversary sees $h = H(\text{pwd})$

Let's model H as a random oracle:
Each time $\mathcal{A}$ wants to evaluate H, it queries an "oracle" that holds a random function.
Because it is a random function, a wrong query reveals nothing about pwd.
Since there are no random oracles in the sky, we then *instantiate* with some hash function.

Is this a reasonable assumption?

- We know that there exist constructions of various cryptographic primitives (e.g. encryption schemes) that are secure in the RO model, but are provably *insecure, regardless of what hash function is used in the instantiation.*
  - However, these constructions are contrived.
- We have seen a MAC scheme that was secure in the RO, but insecure if the hash function instantiating the RO followed the Merkle-Damgard transform.

On the other hand...

Random Oracle

Adversary sees $h = H(\text{pwd})$

Let's model H as a random oracle:
Each time $\mathcal{A}$ wants to evaluate H, it queries an "oracle" that holds a random function.
Because it is a random function, a wrong query reveals nothing about pwd.
Since there are no random oracles in the sky, we then *instantiate* with some hash function.

Is this a reasonable assumption?

- We know that there exist constructions of various cryptographic primitives (e.g. encryption schemes) that are secure in the RO model, but are provably *insecure, regardless of what hash function is used in the instantiation.*
  - However, these constructions are contrived.
- We have seen a MAC scheme that was secure in the RO, but insecure if the hash function instantiating the RO followed the Merkle-Damgard transform.

On the other hand...

- A proof in the RO model helps us identify a single, potential weak point.

Random Oracle

Adversary sees $h = H(\text{pwd})$

Let's model H as a random oracle:
Each time $\mathcal{A}$ wants to evaluate H, it queries an "oracle" that holds a random function.
Because it is a random function, a wrong query reveals nothing about pwd.
Since there are no random oracles in the sky, we then *instantiate* with some hash function.

Is this a reasonable assumption?

- We know that there exist constructions of various cryptographic primitives (e.g. encryption schemes) that are secure in the RO model, but are provably *insecure, regardless of what hash function is used in the instantiation.*
  - However, these constructions are contrived.
- We have seen a MAC scheme that was secure in the RO, but insecure if the hash function instantiating the RO followed the Merkle-Damgard transform.

On the other hand...

- A proof in the RO model helps us identify a single, potential weak point.
- It can be viewed as a proof that the "only" thing that can go wrong is the choice of hash function.