Crash Course in Reductionist Cryptography

Adam Stubblefield
Designing Security Systems

What is Cryptography?

What is Cryptography?

What is Cryptography?

What is Cryptography?

What do we need?

- An *encapsulation* method shared by the two parties (and the adversary)
  - This varies depending on what goal we're trying to achieve
- Some secret information known only to the two parties called the *key*

$$k = 011010$$

$$|k| = 6$$

The Adversary

- The adversary wants to break the security of our encapsulation method
- He isn't all powerful - he's just some (possibly randomized) computer algorithm
- We will say that the system is secure if this bounded adversary can't break our scheme in a reasonable amount of time

Atomic Primitives

- We can't prove that they exist, we have to *assume* that they do
- Moreover, we have to assume that *specific algorithms* implement them
- Fortunately, if one algorithm turns out not to implement one, we can just switch it out for another

Functions

Domain
Inputs

Range
Outputs

$$f : D \rightarrow R$$

Functions Families

- $F : K \times D \to R$
- For each key in K, you get a different function $F_K : D \to R$
- You can also think of it as a multivariable function: $F(k, x) = y$

Let $D, R \subseteq \{0,1\}^*$ be finite non-empty sets. We denote the set of all functions

$$f : D \rightarrow R$$

as $\mathrm{Func}(D,R)$

If $D=\{0,1\}^n$ and

$R=\{0,1\}^m$ we set

$\mathrm{Func}(n,m)=\mathrm{Func}(D,R)$

and

$\mathrm{Func}(n) = \mathrm{Func}(D, D)$

Naming Functions

- If we order the domain $D = (x_1, x_2, ...)$, then we can "name" each function by the values $(f(x_1), f(x_2), ...)$
- We can then create a family of functions out of Func(D, R) by using these names as the keys

Function from Func(3,2)

x	000	001	010	011	100	101	110	111
$f(x)$	01	00	10	11	10	10	01	00

k = (01, 00, 10, 11, 10, 10, 01, 00)

Random Functions

- To select a random function f from this family, just pick a key k at uniformly at random and set $f = F_k$
- Note that this definition of a random function has nothing to do with the function itself and only to do with how it is chosen

Another View

- Think of the random function as a black box
- You can give it an input and it will give you the corresponding output:
  - 101? 10. 111? 00. 101? 10.
- It always has to give you the same output when you repeat an input

As a Program

```
Function f(x):
If I've been asked about x before
        Return t[x]
Else
        Set t[x] to a random element of the range
        Return t[x]
```

Fix $X = \{0, 1\}^n$

and $Y = \{0,1\}^m$, then

$\Pr[f(X)=Y] =$

Fix $X = \{0, 1\}^n$

and $Y = \{0,1\}^m$, then

$$\Pr[f(X)=Y] = \frac{1}{2^m}$$

Fix $X_1, X_2 = \{0, 1\}^n$

and $Y = \{0, 1\}^m$, then

$\Pr[f(X_1)=Y | f(X_2)=Y] =$

Fix $X_1, X_2 = \{0, 1\}^n$

and $Y = \{0, 1\}^m$, then

$$\Pr[f(X_1)=Y|f(X_2)=Y] = \frac{1}{2^m}$$

$$\Pr[f(X_1)=Y \text{ and } f(X_2)=Y] =$$

If $X_1 = X_2$

If $X_1 \neq X_2$

$$\Pr[f(X_1)=Y \text{ and } f(X_2)=Y] =$$

$$\frac{1}{2^m} \quad \text{If } X_1 = X_2$$

$$\text{If } X_1 \neq X_2$$

$$\Pr[f(X_1)=Y \text{ and } f(X_2)=Y] =$$

$$\frac{1}{2^m} \qquad \text{If } X_1 = X_2$$

$$\frac{1}{2^{2m}} \qquad \text{If } X_1 \neq X_2$$

$$\Pr[f(X_1) \oplus f(X_2)=Y] =$$

If $X_1 = X_2$ and $Y = 0$

If $X_1 = X_2$ and $Y \neq 0$

If $X_1 \neq X_2$

$$\Pr[f(X_1) \oplus f(X_2) = Y] = \begin{cases} 1 & \text{If } X_1 = X_2 \text{ and } Y = 0 \\ 0 & \text{If } X_1 = X_2 \text{ and } Y \neq 0 \\ \dfrac{1}{2^m} & \text{If } X_1 \neq X_2 \end{cases}$$

Pseudorandom Function

- Informally, a *pseudorandom function* (PRF) is a family of functions whose members are difficult for an adversary to distinguish from a random function

Pseudorandom Function

- We're going to give the adversary *oracle access* to a function *g*
  - He can ask what *g* returns given any inputs
- Sometimes *g* will be a randomly selected from our pseudorandom family, sometimes *g* will be a random function
- The adversary will try to tell us which *g* is

World 0

Random Function

g

World 1

Pseudorandom Function

g

I’m in world 0

World 0

Random Function

g

World 1

Pseudorandom Function

g

I’m in world 0

More Formally

- We want to quantify how good an adversary A is at telling world 0 from world 1
- We call this the *advantage* of adversary A, and compute it:

Pr[A says 1 in world 1] - Pr[A says 1 in world 0]

Adversaries

- Different adversaries have different advantages
- Some adversaries might just be more "clever" than others
- Some adversaries might use more resources that others

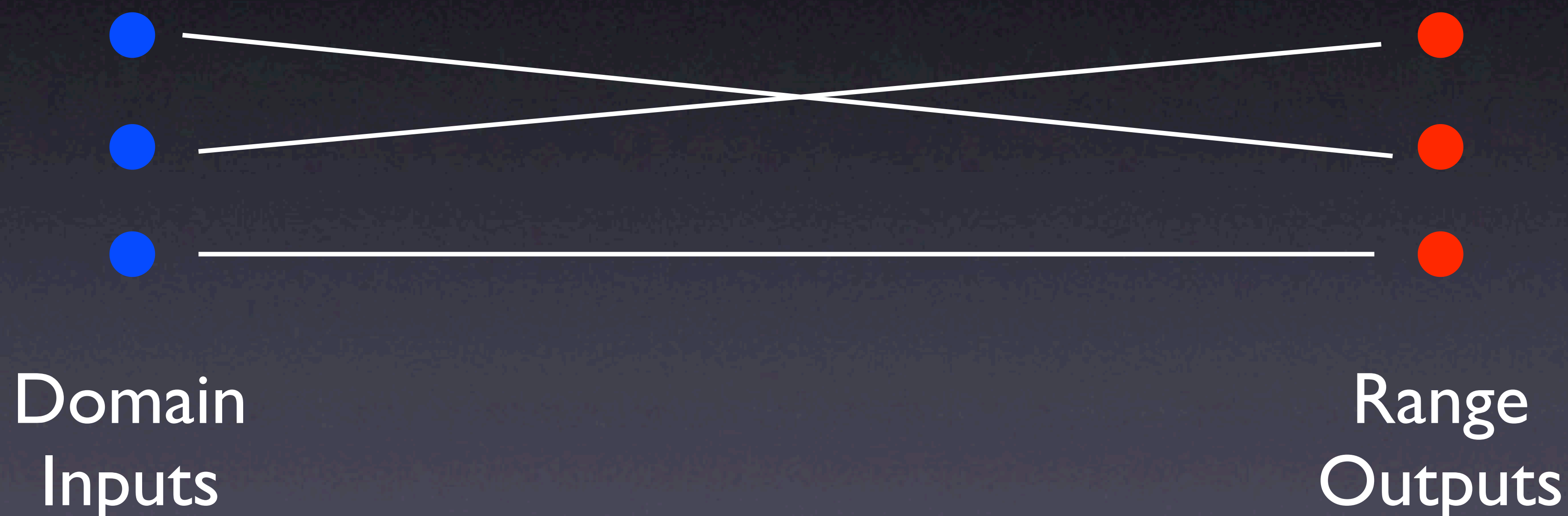
Resources

- Time: what is the running time (computational complexity) of A?
  - Also includes the size of A's code and the running time of setting up the worlds
- Queries: how many times does A query the g oracle?

Security of a PRF

- A PRF *F* is “secure” if all “reasonable” adversaries have “small” prf-advantage
- The prf-advantage of all (t,q)-bounded adversaries in distinguishing *F* is less than ϵ

Permutations



$$f : D \to D$$

Let $D \subseteq \{0,1\}^*$ be finite non-empty sets and let n, N ≥ 1 be integers. We denote the set of all functions

$$f : D \to D$$

as Perm(D)

If $D=\{0,1\}^n$ we set

$\mathrm{Perm}(n) = \mathrm{Perm}(D)$

Random Permutation

- You can key the permutations just like the functions, and select a random permutation by selecting a random key
- The algorithmic definition is a little different: you have to make sure that you never reuse an element in the range

Notions of Security For Pseudorandom Permutations (PRP)

- Chosen Plaintext Attack (CPA): attacker has to decide whether *g* is a random permutation or a PRP
- Chosen CIphertext Attack (CCA): attacker also gets access to the inverse of *g*

Next Time

- How do we prove things using these definitions?
- Why are PRFs and PRPs important to me?