Random Oracles, Revisited

Ran Canetti, Oded Goldreich,
Shai Halevi

Provable Security

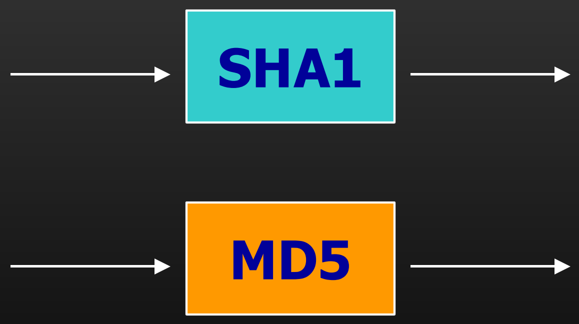
- We want to **prove** that a crypto scheme is secure before we use it

Problem:

- What if our crypto scheme uses elements that aren’t provably secure?

Example: Hash Functions

- Real world examples: SHA1, MD5
- We would like a hash function that has:
  - Perfect randomness
  - Collision resistant
  - One-way

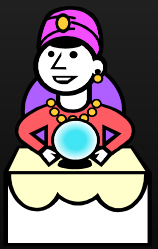


Example: Hash Functions

- But what if these things aren’t enough?
- Sometimes we don’t know even know what features we need!

Recall: The Random Oracle Model

- Random Oracle Model (ROM) lets us replace un-provable components with “perfect” oracles
- With these taken care of, we can prove the rest of the scheme



But…

- When we implement the ideal system in the real world, where random oracles don't exist, will it still be secure?

Not necessarily!

- We will come up with crypto schemes where the real world implementation will **always** be insecure
  - **no matter what** function or collection of functions we replace the oracle with
  - Yet they will be provably secure in the Random Oracle Model

Uh oh…

- If the Random Oracle Model says something’s **secure**
- But we know it’s **insecure** in the real world…
- Can we trust the Random Oracle Model?

Our Ingredients

- To demonstrate this problem, we need to define some concepts:
  - Signature Schemes
  - Relations and Evasive Relations
  - Correlation Intractability

Signature Schemes

- Signatures are a little bit like MACs
- When we sign a message, we can:
  - Authenticate messages
  - Prove authorship
  - Prevent tampering

What is a Signature Scheme?

- Unlike MACs, signatures are based on Public keys and Private (secret) keys
  - **Signer** generates a signature with Secret key
  - **Verifier** checks the signature with Public key and **accepts** or **rejects** it

Anyone can Verify

- The **Verifier** can be anyone
  - We give out our Public Key
  - Anyone can check a signature (even bad guys)

Security of Signature Schemes

- A signature scheme is **secure** if:
  1. It's **easy** to sign
  2. It's **hard** for an adversary to forge signatures the signer didn't generate

Easy and Hard

- **Easy** for the signer to sign means
  - Signing takes a polynomial number of time steps

Polynomial Time = O(poly(k))
k = number of bits
poly() = some polynomial

Easy and Hard

- **Hard** to forge means
  - The probability an adversary can generate forged signatures is **negligible**
  - This also means the adversary should never recover the secret key!

Relations

- A **relation** is any set of conditions by which we can evaluate some pair (x, y)
  - {for all x, x=y}
  - {for all x, x≠y}
  - {for all y, x is an odd number}
  - {for all x, y>100 AND y<100}

Satisfying a Relation

- A Relation is **satisfied** by a pair (x, y) if the pair **meets its conditions**
- For instance, we could require that y=f(x) for some function f
- Let's give some examples…

Example #1

R = {x==y} (equality)

f(x)=2x

x=0, f(0)=0 (satisfies R)

x=1, f(1)=2

x=2, f(2)=4

x=3, f(3)=6

x=4, f(4)=8

…

Evaluating Relations

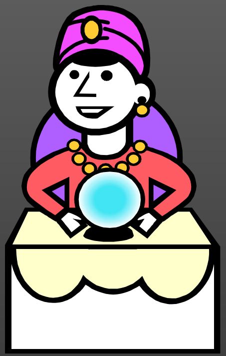
- We want relations that can be **evaluated** in a reasonable amount of time
  - In other words, we need an **efficient** algorithm to check if (x, y) satisfies the relation

Rare or Evasive Relations

- When we’re looking at the input-output pairs of a random oracle, some relations are more likely to be satisfied.
  - {for all x, x=y} (very rare)
  - {for all x, x≠y} (very common)
  - {for all y, x is an odd number} (50%)
  - {for all x, y>100 AND y<100} (never)

What's an Evasive relation?

query x



O(x)

oracle

- Pick a relation R
- Try to find an x for which (x,O(x)) is in R
- If the probability you can do this is negligible, R is evasive

In other words, *by definition*, an evasive relation is unlikely to be satisfied by the input-output pairs of a random oracle

Example of an Evasive Relation

- R={x, f(x)} for any function f
- The above relation is **evasive** on pairs (x, O(x))
  - For each x, the probability that O(x)=f(x) is **extremely** small

Correlation Intractability

- A property of random oracles that we want our "real world" function to have

Correlation Intractability

- A function f is Correlation Intractable if:
  - For **every** evasive relation, it's difficult to find x such that (x,f(x)) satisfies the relation with non-negligible probability

Correlation Intractability

- Remember: we defined evasive relations as those relations not satisfied by the input-outputs of a random oracle
- So Random Oracles are correlation-intractable
- If all evasive relations are not satisfied by a function f, f is also correlation intractable

There are no correlation intractable functions!

- Simple enough to prove:
  - Pick a relation R={x, f(x)}
  - Take any oracle input/output (x, O(x))
  - The chance that the random value O(x) will equal f(x) for any x is **negligible**
  - But of course, (x, f(x)) will always satisfy the relation

So far…

- We've identified a property of random oracles that functions don't have

Property of random oracles

- We know random oracles are correlation intractable
- And functions are **not** correlation intractable

The Big Picture

- If we can prove a scheme **secure** in the Random Oracle Model…
- And prove it's **insecure** in the real world
- Then there's a problem with the Random Oracle Model !

Our Aim

- We want to build a scheme that's **secure** using Random Oracles
- But insecure using **functions**

Outline of our Proof

- Start with a perfectly good Signature Scheme that's secure

Message

Signer

Signature

Bing

- Add a deliberate "bug" to our scheme
  - If some condition arises, our scheme does something really **insecure**

Message

Signature ☺
or
Secret Key ☹

If condition X
reveal the secret key !!!
Else
sign with the original, secure scheme

Bang

- Show that this condition **won’t occur in the Random Oracle Model**
  - In the R.O.M., our scheme is **secure**

Message →

→ **Signature ☺ or Secret Key ☹**

If condition X
reveal the secret key !!!
Else
➔ **sign with the original, secure scheme**

Bongo

- Show that the condition **will occur in the “real world”** when we replace Random Oracles with functions
  - So implementation is **insecure**

Message

Signature ☺

or

Secret Key ☹

If condition X

reveal the secret key !!!

Else

sign with the original, secure scheme

So what’s the trick…

- The trick to all of this is figuring out what to use for “Condition X”
  - Something that **won’t** happen in the Random Oracle Model…
  - But **will sometimes** happen with functions

Adding a condition

- So the scheme in ROM must be different from the scheme in the real world
- So far, there is no difference
- So let's add a random oracle to our scheme in ROM, which is replaced by a function in the real world

Adding a Random Oracle

Message →  → Signature ☺ or Secret Key ☹

If **[some condition involving an oracle]**
reveal the secret key !!!
Else
sign with the original, secure scheme

Adding a condition

- So our condition is going to involve a random oracle
- We'll need something that can tell the difference between a random oracle (in ROM) and a function (in the real world)
- But we know of something that can do this!

We need an evasive relation

- An evasive relation will never be satisfied by a random oracle, but can be satisfied by a function
- So we'll pick some evasive relation
- It should be something that our adversary will know how to satisfy

Pick our Evasive Relation

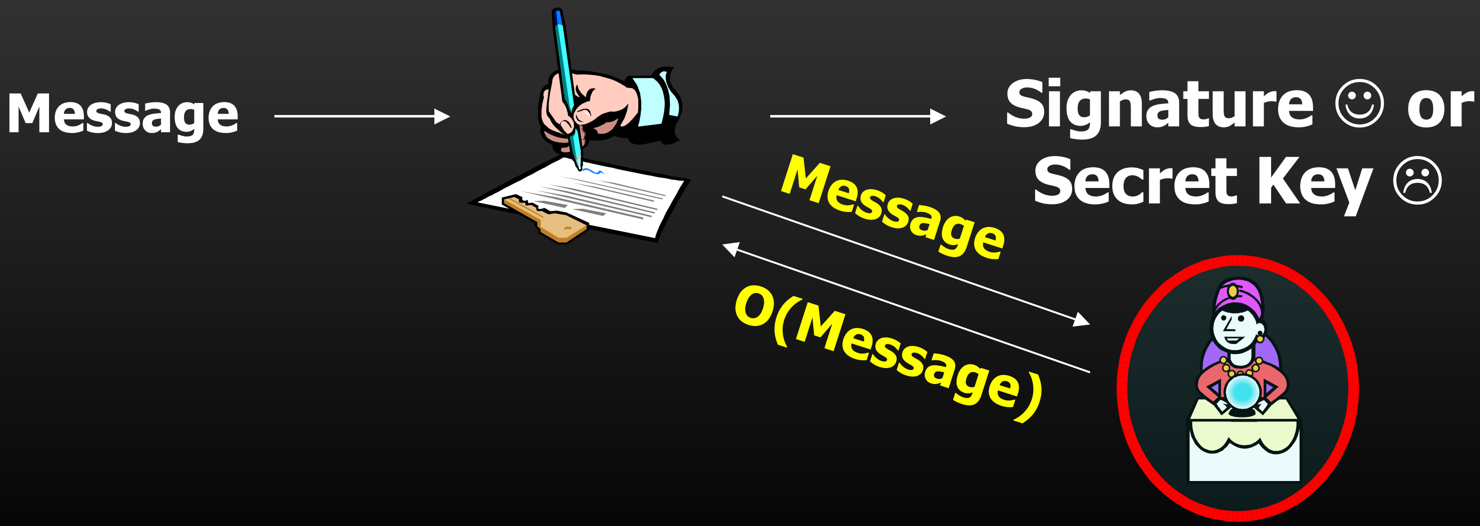
- How about the relation R= { x,f(x) } ?
- It's evasive
- And it's easy to satisfy in the real world if f is the same function used to replace the oracle

And our condition is…

Pick some evasive relation R

R = { x, f(x) }

If [message, O(message) satisfies R]
reveal the secret key !!!
Else
sign with the original, secure scheme

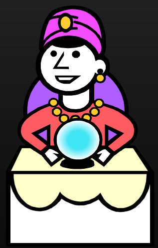


The scheme we've built

- Now that we have our condition and our evasive relation, we have everything we need for our scheme
- Let's go through our entire scheme now, step by step

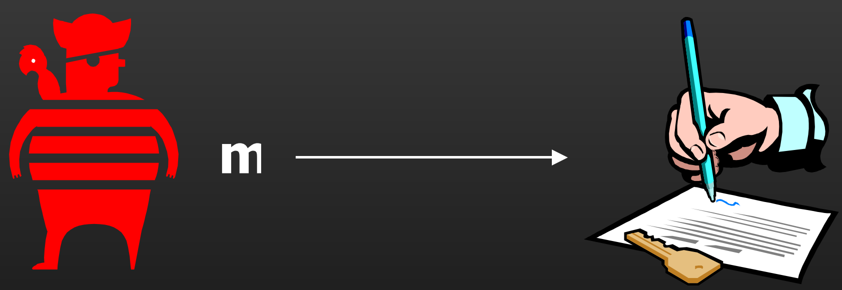
Our Scheme in ROM

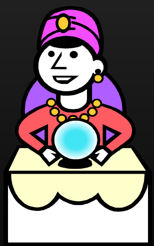
- Pick $R = \{ x, f(x) \}$ as our evasive relation



Our Scheme in ROM

- Adversary sends message m to the Signer

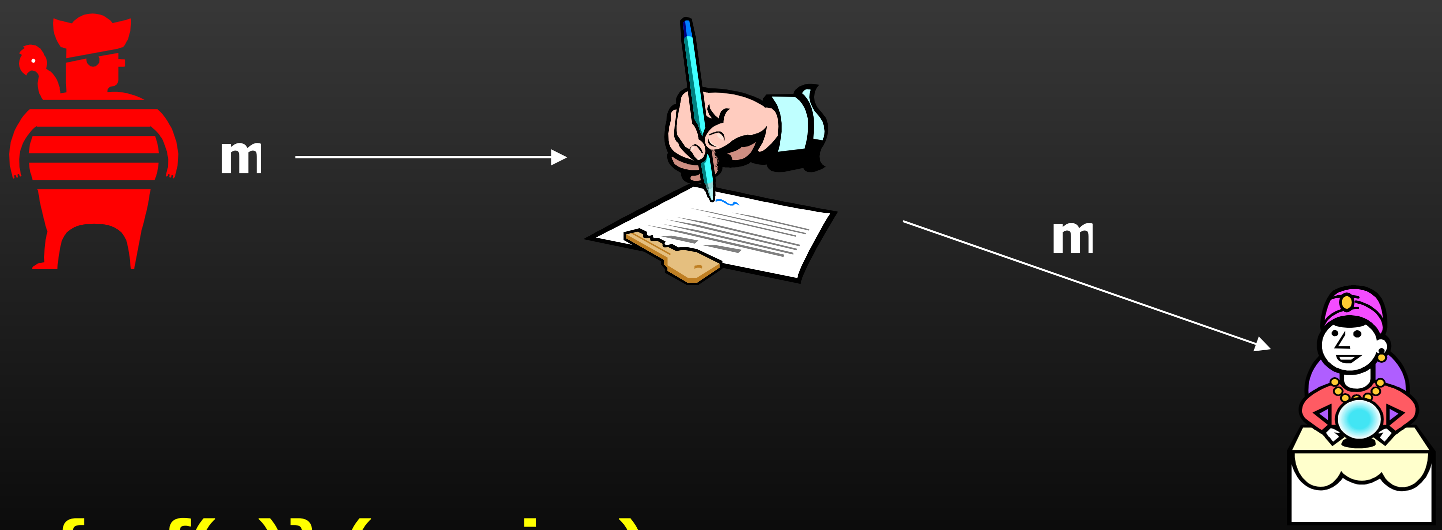




R = {x, f(x)} (evasive)

Our Scheme in ROM

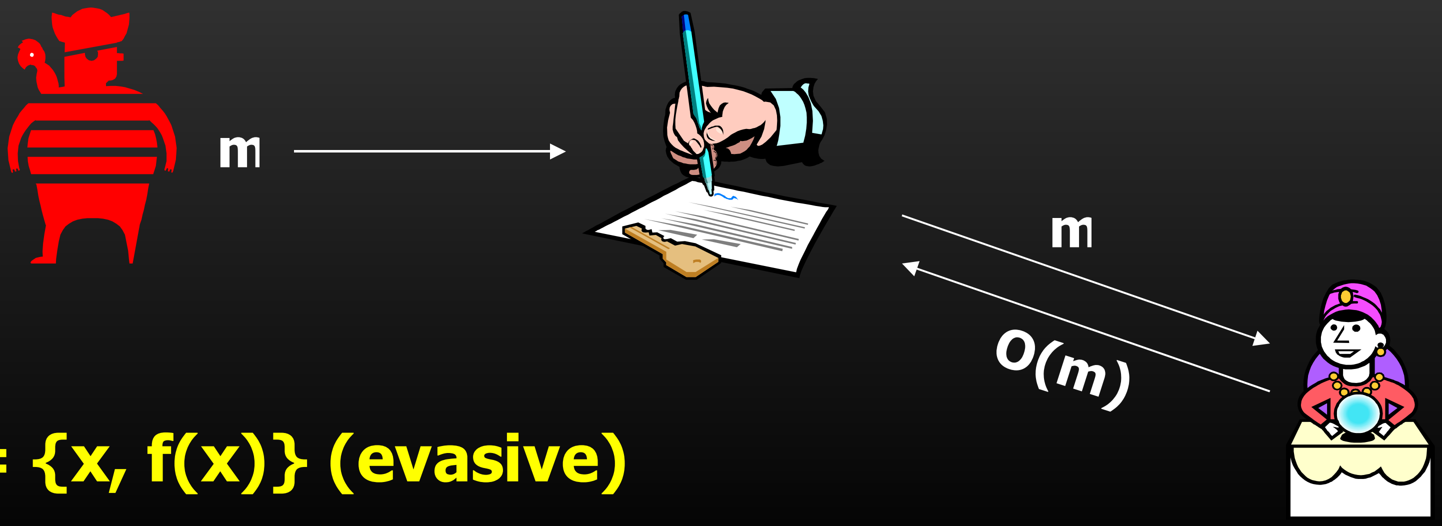
- Adversary sends message m to the Signer
- Signer sends m to Random Oracle



R = {x, f(x)} (evasive)

Our Scheme in ROM

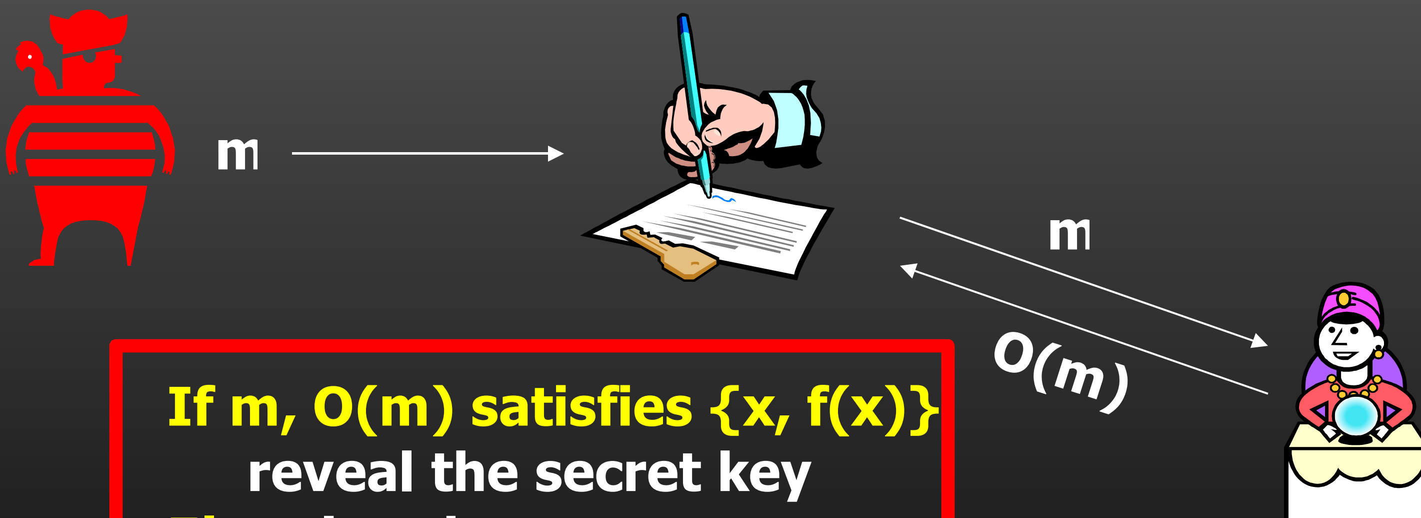
- Adversary sends message m to the Signer
- Signer sends m to Random Oracle
- Signer gets O(m) from Random Oracle



R = {x, f(x)} (evasive)

Our Scheme in ROM

- Signer checks if (m, O(m)) satisfies R

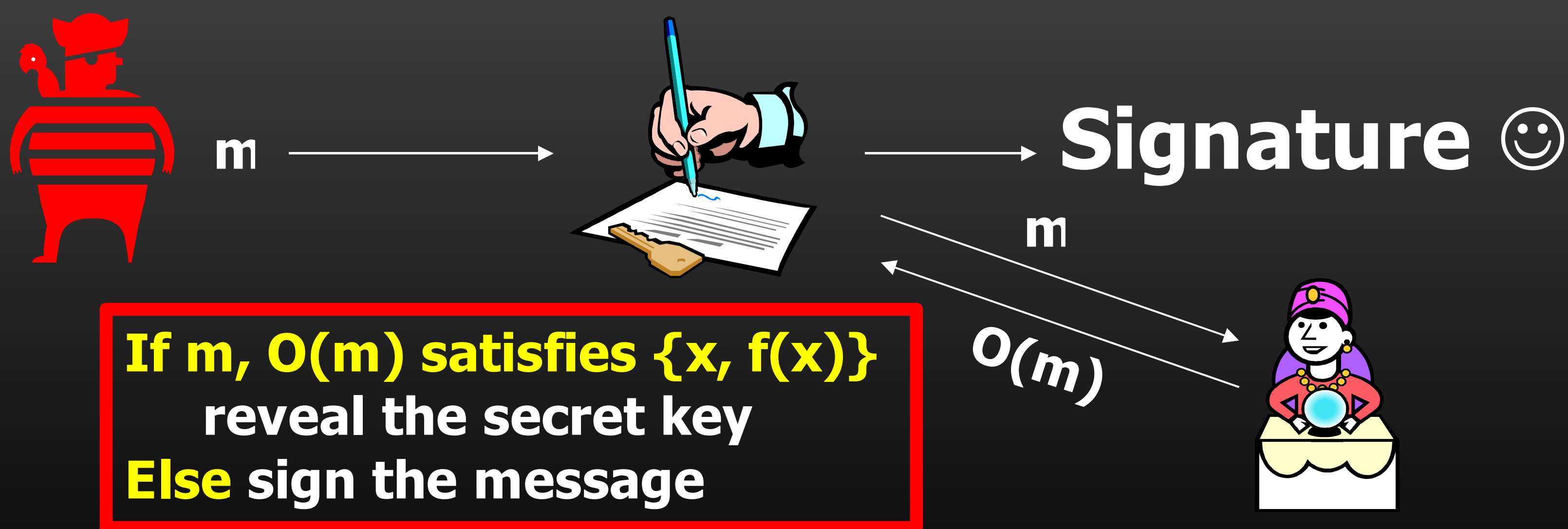


If m, O(m) satisfies {x, f(x)}
reveal the secret key
Else sign the message

R = {x, f(x)} (evasive)

Our Scheme in ROM

- The output of a Random Oracle will practically **never** satisfy an evasive relation



R = {x, f(x)} (evasive)

This scheme is secure in ROM

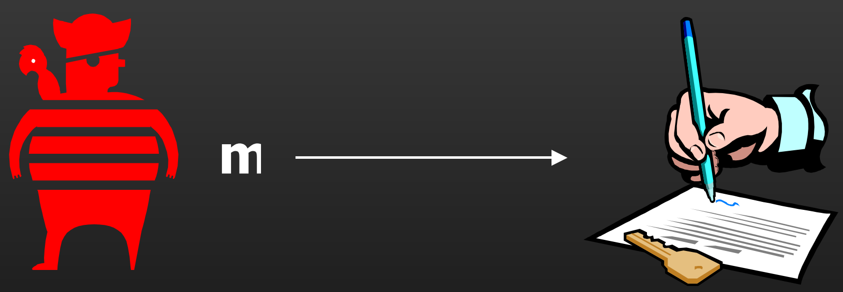
- In the Random Oracle Model, our condition is never met
- The scheme never reveals its secret key
- We've shown that it is **secure** in the ROM

Now, in the real world

- When we **implement** the scheme, we replace the Random Oracle
- Let’s replace it with a function **f**

Our Scheme in the Real World

- Adversary sends message m to the Signer

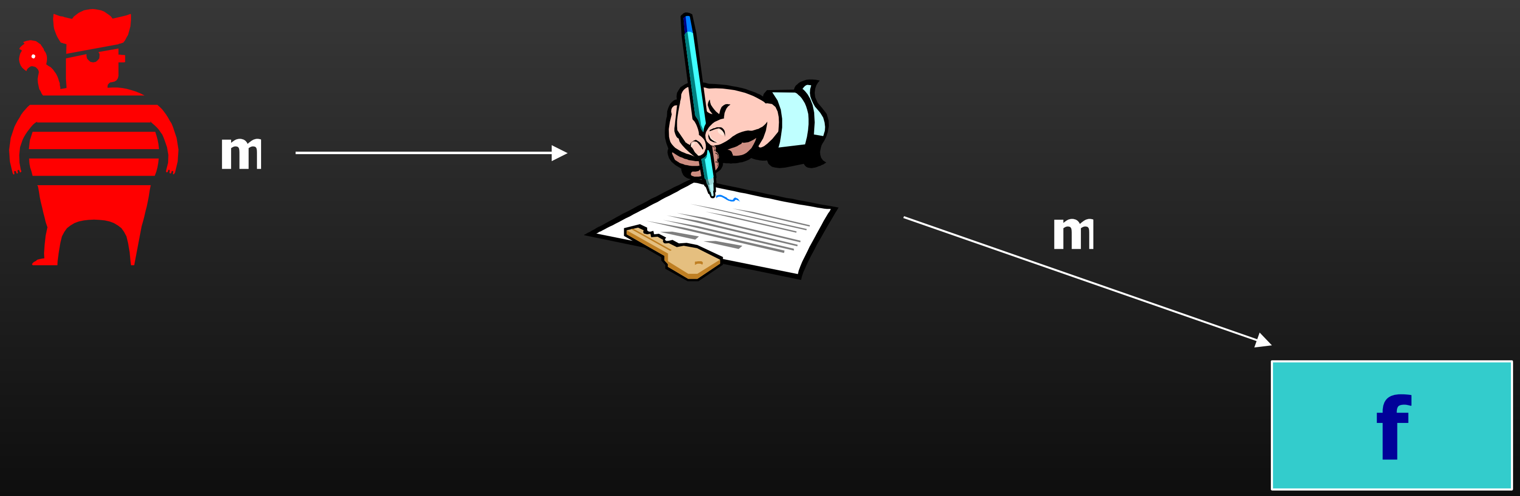


R = {x, f(x)} (evasive)

f

Our Scheme in the Real World

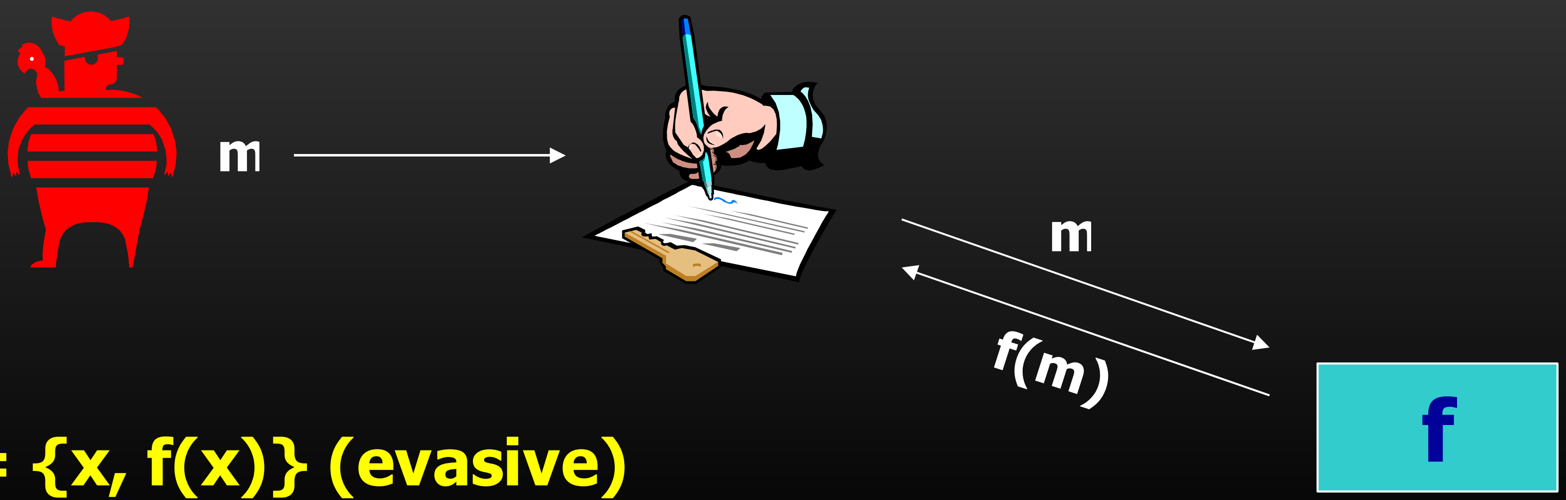
- Adversary sends message m to the Signer
- Signer sends m to function f



R = {x, f(x)} (evasive)

Our Scheme in the Real World

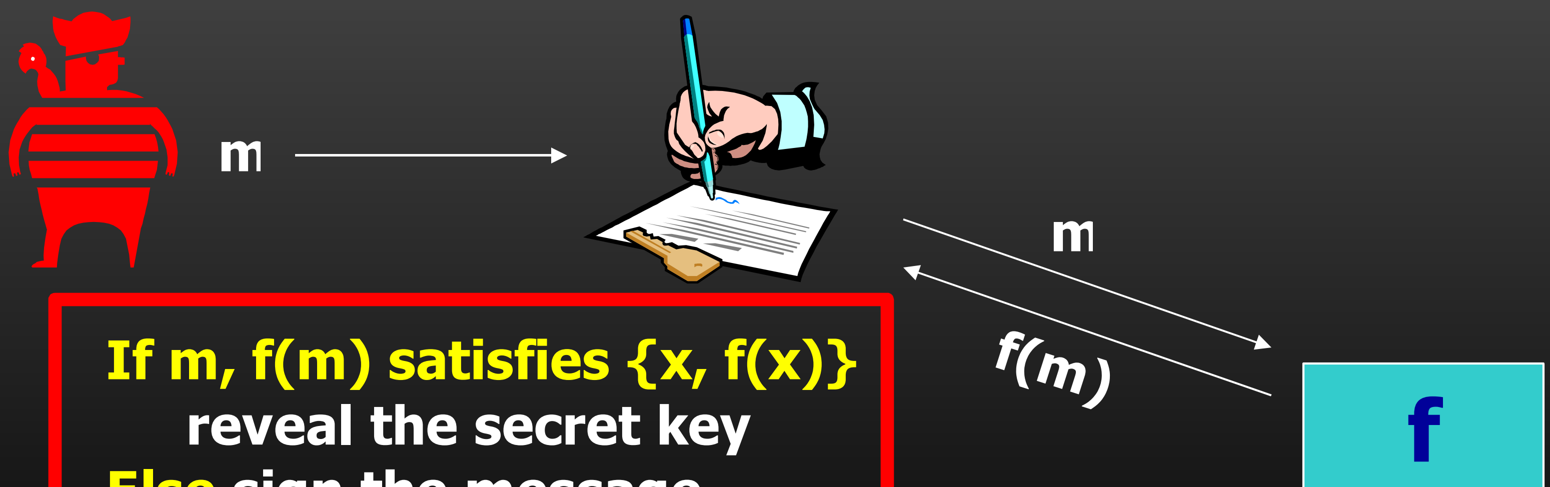
- Adversary sends message m to the Signer
- Signer sends m to function f
- Signer gets f(m) from function f



R = {x, f(x)} (evasive)

Our Scheme in the Real World

- Signer checks if (m, f(m)) satisfies R

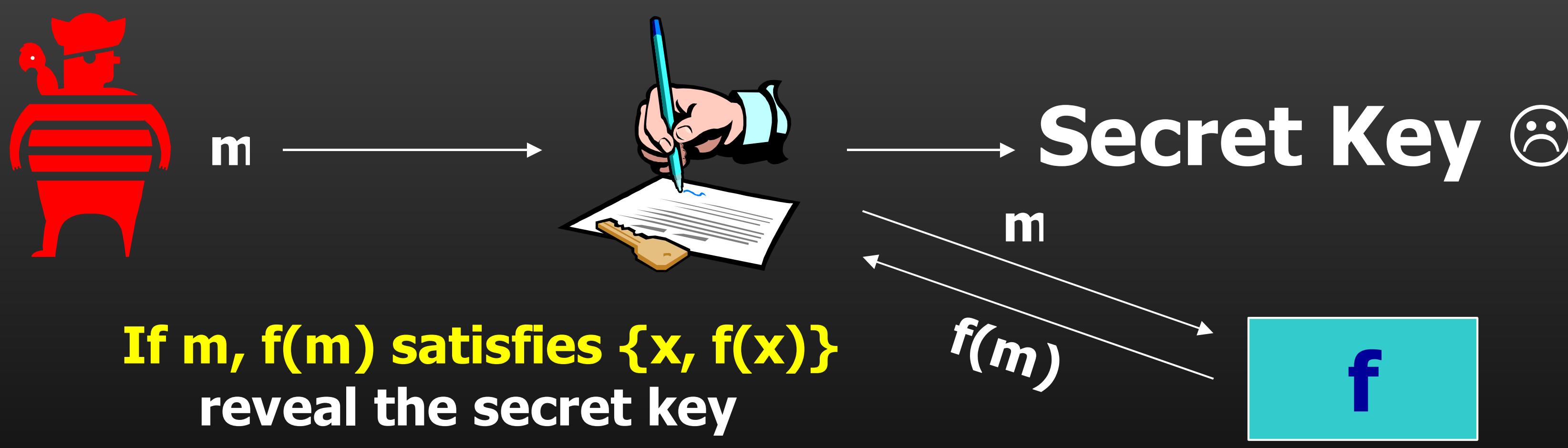


If m, f(m) satisfies {x, f(x)}
reveal the secret key
Else sign the message

R = {x, f(x)} (evasive)

Our Scheme in the Real World

- The relation **is** satisfied, so the signer reveals its secret key



If m, f(m) satisfies {x, f(x)}
reveal the secret key
Else sign the message

R = {x, f(x)} (evasive)

This scheme is not secure!

- An adversary can get this scheme to reveal its secret key
- In fact, because of what we picked for R, the adversary can always break the scheme
- It's definitely **insecure** in the real world

So far…

- Random Oracles have properties that functions don't
- We leveraged this to build a scheme **secure** in the ROM, **insecure** with a function

Another Attempt

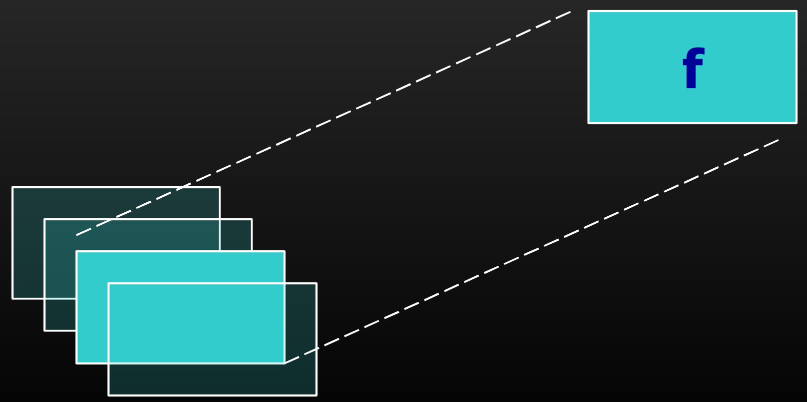
- Maybe using one function is too easy
- What if we implement using a collection of functions (aka "**function ensemble**")
  - Ensemble is a collection of of functions $f_1..f_n$
  - Like using a keyed hash, or MAC

Are Function Ensembles Better?

- We will show that **function ensembles** have the same problem as **functions**
- **We** can still build schemes that are **secure** in the Random Oracle Model, but **insecure** with a function ensemble

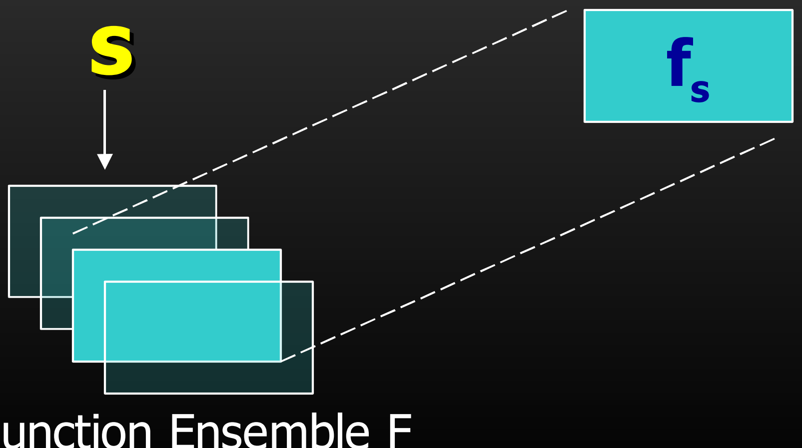
Using a Collection of Functions

- It’s great to have a collection of functions, but we can only use **one** at a time
- Everyone participating in the scheme must know what function we’re using



Using a Function Ensemble

- To use a function ensemble:
  - Select one function f_s at random when we start our scheme, and tell everyone what **s** is

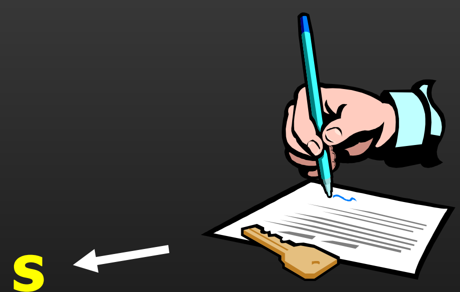


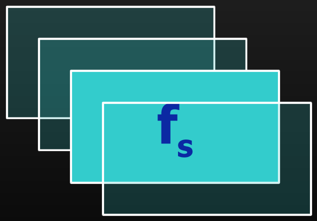
The Proof

- Again we choose an evasive relation R
- This time $\mathbf{R=\{x,f_x(x)\}}$
- We know this evasive relation can’t be satisfied by a Random Oracle

Implementation with Function Ensembles (Setup)

- First: signer picks one function f_s from a function ensemble
- And publishes s to everyone





Function Ensemble F

The adversary attacks

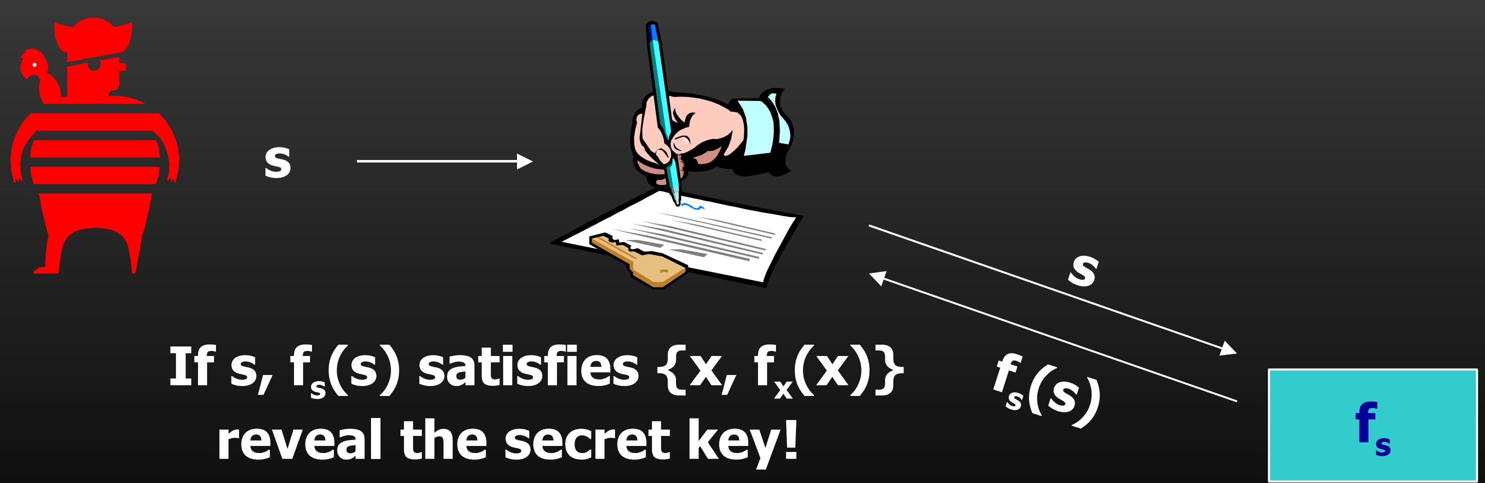
- Adversary submits **s** as the message

If s, $f_s(s)$ satisfies {x, $f_x(x)$}
reveal the secret key!
Else
sign with the original, secure scheme

f_s

Signer calls Function Oracle

- Signer gets $f_s(s)$



If s, $f_s(s)$ satisfies {x, $f_x(x)$}
reveal the secret key!
Else
sign with the original, secure scheme

The adversary attacks

- Signer checks if s, $f_s(s)$ satisfies the relation $R = \{x, f_x(x)\}$

s

s

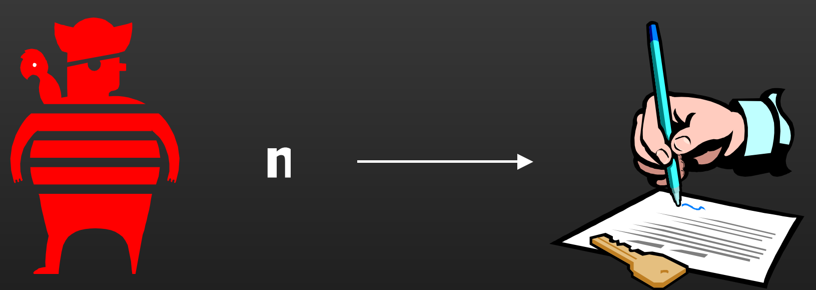
$f_s(s)$

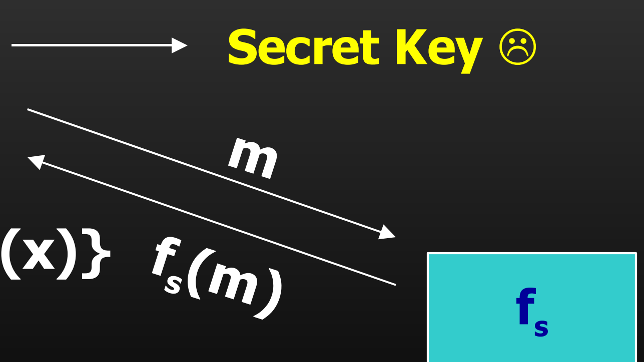
f_s

If s, $f_s(s)$ satisfies $\{x, f_x(x)\}$
reveal the secret key!
Else
sign with the original, secure scheme

The adversary attacks

- The relation **is** satisfied, so the signer reveals its secret key





If m, f_s(m) satisfies {x, f_x(x)}

➡ **reveal the secret key!**

Else

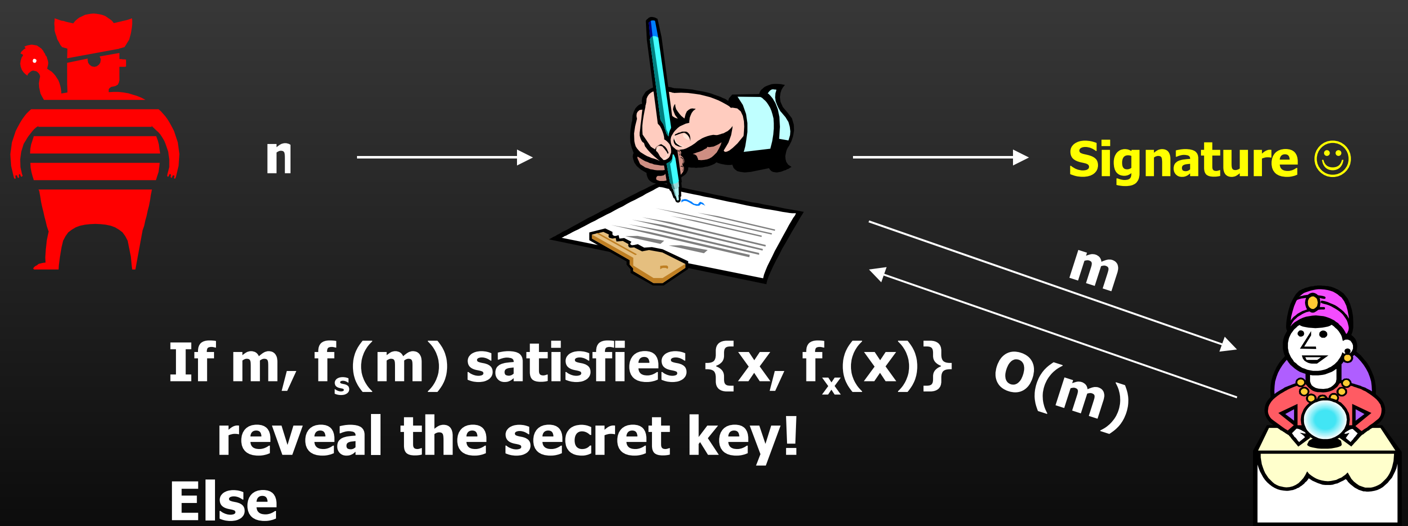
sign with the original, secure scheme

Thus…

- The scheme is insecure when implemented with a **function ensemble**

But with Random Oracles…

- The relation will **not** be satisfied by a Random Oracle
- So it will always sign **securely**



If m, f_s(m) satisfies {x, f_x(x)}
reveal the secret key!
Else
➔ **sign with the original, secure scheme**

So…

- As before, the proof works in the Random Oracle Model
- But it doesn’t work if we use function ensembles

There's a small problem

- We've shown that we can
  1. pick **one** function ensemble F
  2. Build a scheme that's **insecure** if we implement it using F

Problem

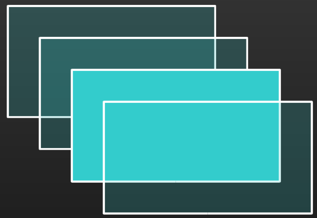
- We built our scheme to work with F
- So our proof **only** holds with F
- Somebody could implement our scheme with F'
  - And our proof might not hold anymore

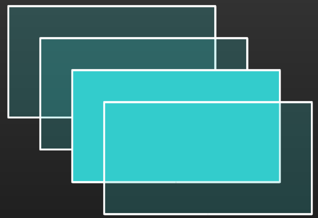
To fix this

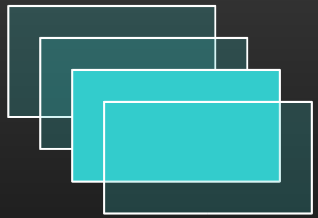
- We must rig our scheme so that it'll be **insecure** for **every possible** function ensemble in the universe

Collections of Ensembles

- We start with a collection of function ensembles
  - Chosen from the collection of all function ensembles

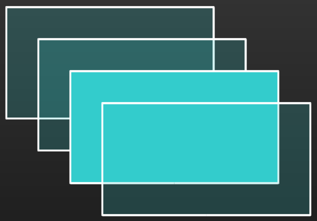

Function Ensemble F1


Function Ensemble F2


Function Ensemble F3

Collections of Ensembles

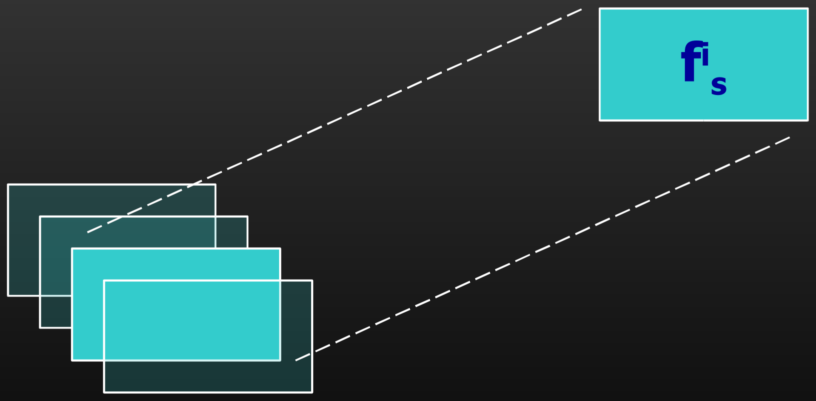
- When we start our scheme, we pick the **i**th function ensemble at random



Function Ensemble F_i

Collections of Ensembles

- From ensemble F_i, we pick the $\mathbf{s}^{th}$ function at random
- This gives us our function $\mathbf{f^i_s}$



Function Ensemble F_i

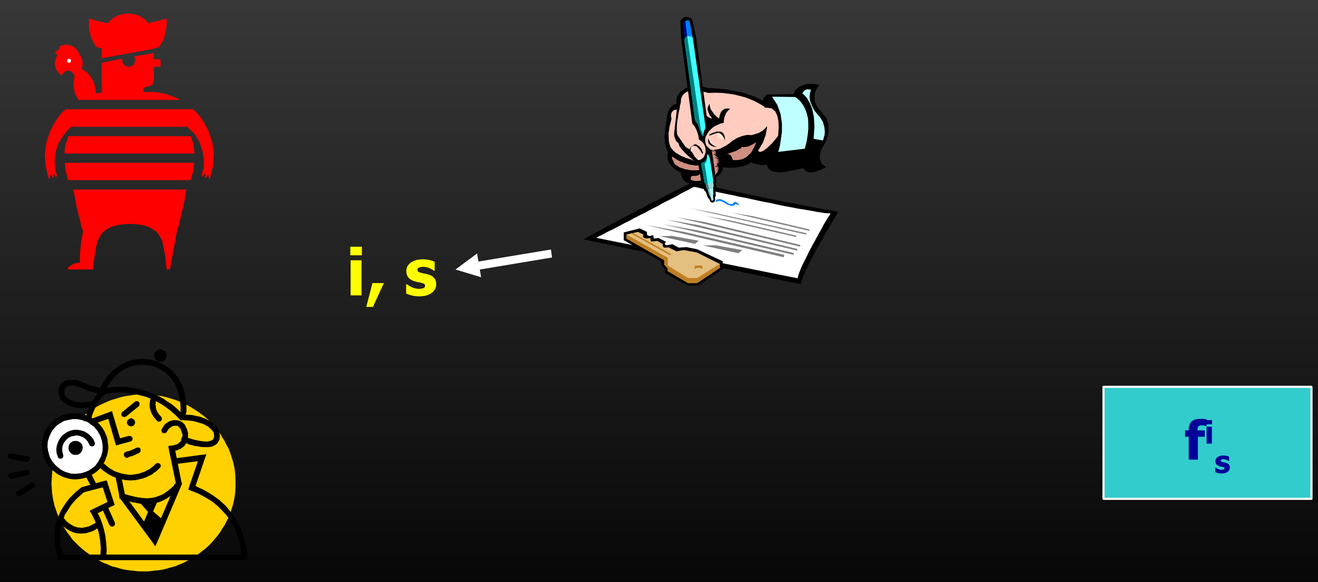
Same Idea

- The proof is familiar
- First, pick an evasive relation:

  This time $R=\{x,f^i_s(x)\}$
- Again, we know this evasive relation won't be satisfied by a Random Oracle

Implementation with Many Function Ensembles (Setup)

- Signer has chosen ensemble $\mathbf{F_i}$, then function f^i_s from it
- And publishes (i, s) to everyone



The adversary attacks

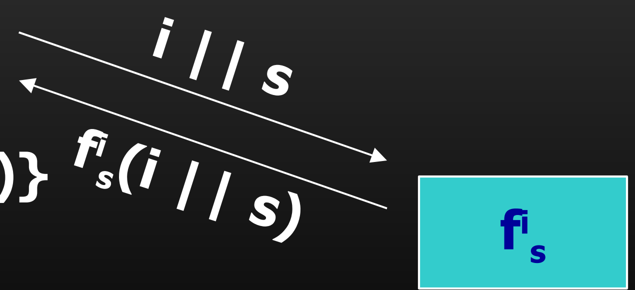
- Adversary submits **i || s** as the message

If i || s, f^i_s(i || s) satisfies {x, f^i_s(x)}
reveal the secret key!
Else
sign with the original, secure scheme

f^i_s

Signer calls the Function Oracle

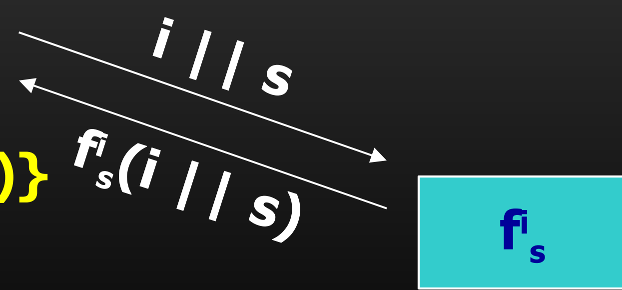
- Signer calls the oracle to get $f^{i}_{s}(i \,||\, s)$



If $i \,||\, s$, $f^{i}_{s}(i \,||\, s)$ satisfies $\{x, f^{i}_{s}(x)\}$
reveal the secret key!
Else
sign with the original, secure scheme

Signer checks the result

- Signer checks if $i \,||\, s$, $f^i_s(i \,||\, s)$ satisfies the relation $R = \{x, f^i_s(x)\}$



If $i \,||\, s$, $f^i_s(i \,||\, s)$ satisfies $\{x, f^i_s(x)\}$

reveal the secret key!

Else

sign with the original, secure scheme

Scheme breaks

- The relation **is** satisfied, so the signer reveals its secret key

If s, f^i_s(i || s) satisfies {x, f^i_s(x)}

➔ **reveal the secret key!**

Else

sign with the original, secure scheme

Again…

- The scheme is insecure when implemented with any **function ensemble**

But with Random Oracles…

- Same as before, the evasive relation will never be satisfied, so the scheme will be **secure**

Running time of the scheme

- We have one problem:
  - Signature schemes must take polynomial time
  - In other words, we need to define a single polynomial that **bounds** the running time of our scheme

Running time of the scheme

- Unfortunately:
  - All of the functions we consider take all polynomial time to evaluate
  - But we have to consider every possible one
- We can’t come up with one polynomial that bounds an infinite number of functions

Running time of the scheme

- The best we say is that our scheme runs in at most **super-polynomial** time
- That violates the requirements of a signature scheme

Stage 3: Reducing the Time

- In this final step we make our Signer and Verifier run in polynomial time, by using something called "Computationally Sound proofs" (CS Proofs)

S. Micali. "CS Proofs". 1994.

What's a Computationally Sound (CS) proof?

- A Computationally Sound Proof is a system for generating and verifying proofs of statements
- When we say "proofs", we mean proofs that are computer-generated and computer-verifiable

S. Micali. “CS Proofs”. 1994.

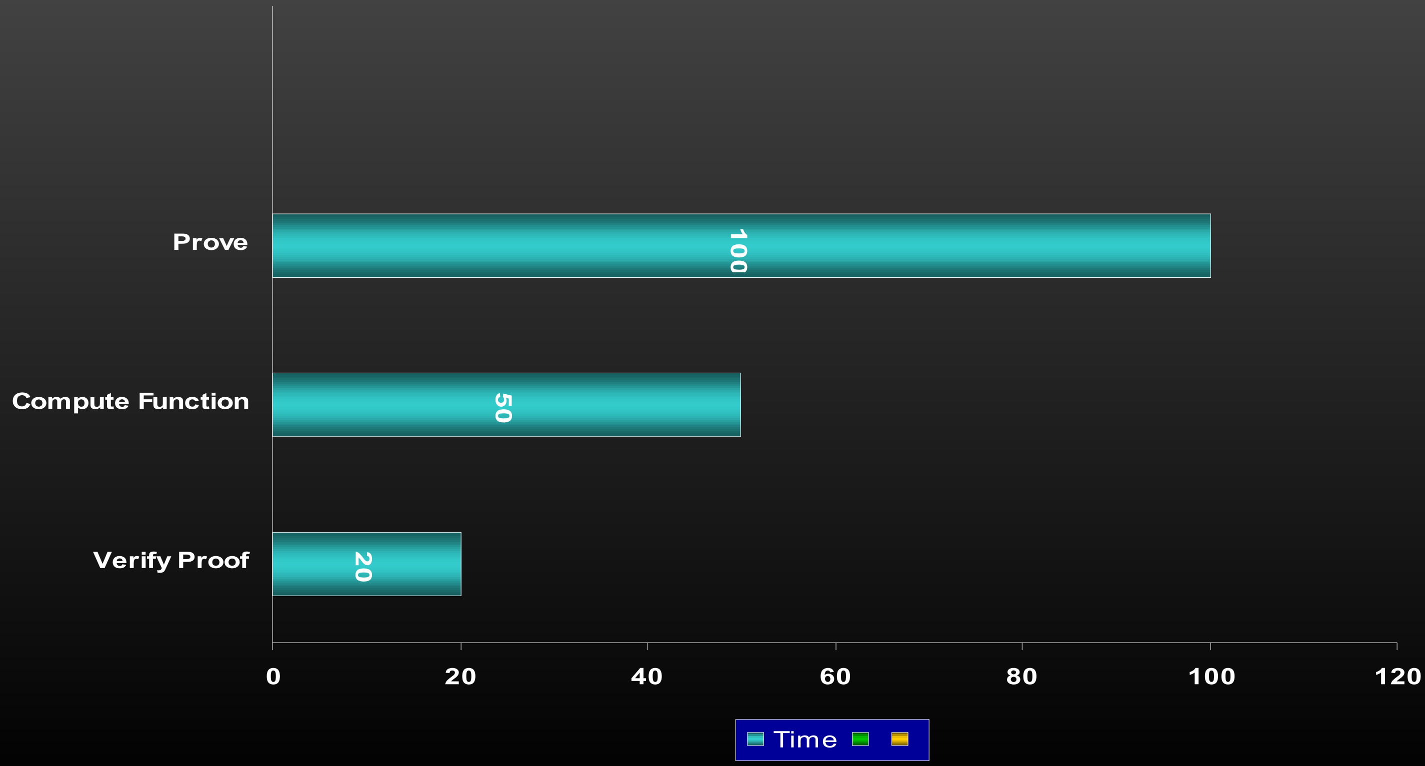
How we use CS Proofs

- Instead of evaluating f, we make the attacker give us a proof of the statement "$(s, f^i_s(s))$ is in R^U"
- We only need to verify that the proof is valid

S. Micali. "CS Proofs". 1994.

Why do we do this?

- Verifying a proof takes less time than computing $f^i_s(x)$



How much less time?

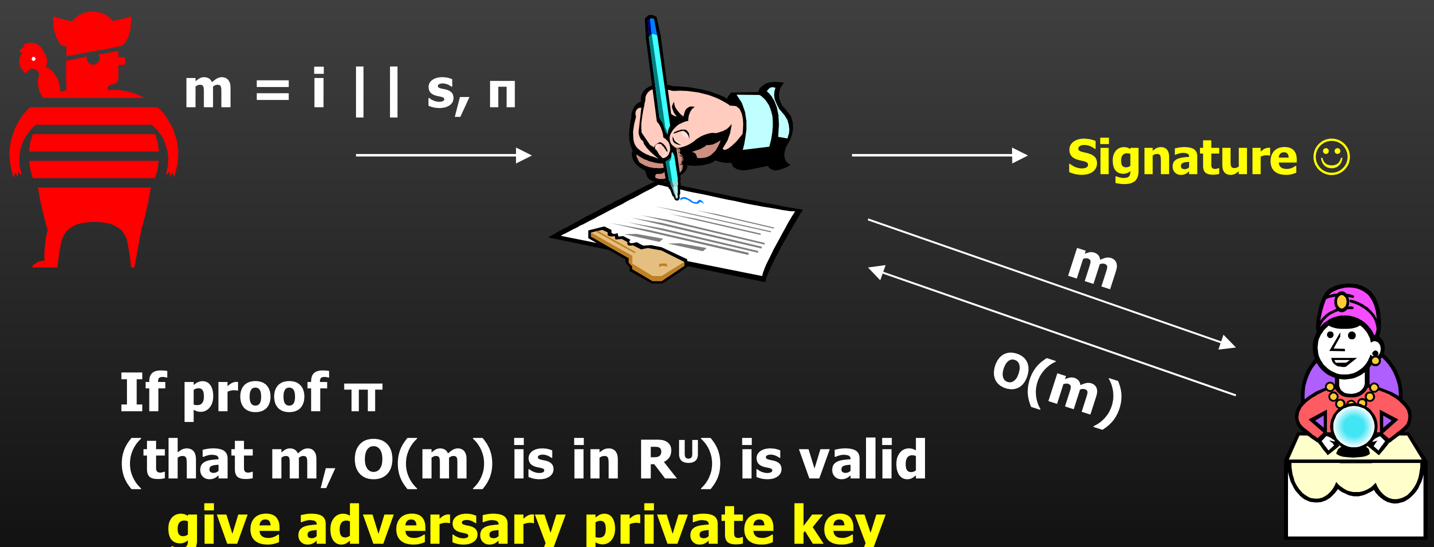
- CS proofs always take sub-polynomial time to verify
- So the time to verify **any** proof is bounded by a single polynomial

Using CS Proofs

- We take our last scheme and modify it
  - The adversary generates a proof π that his input satisfies the relation
  - The signer **verifies** this proof (will always take polynomial time)
  - If the proof is valid, the signer reveals his secret key

Scheme using CS Proofs in ROM

- Pick an evasive relation $R^U = \{\ i \,||\, x\ ,\ f^{i}_{x}(i \,||\, x)\ \}$



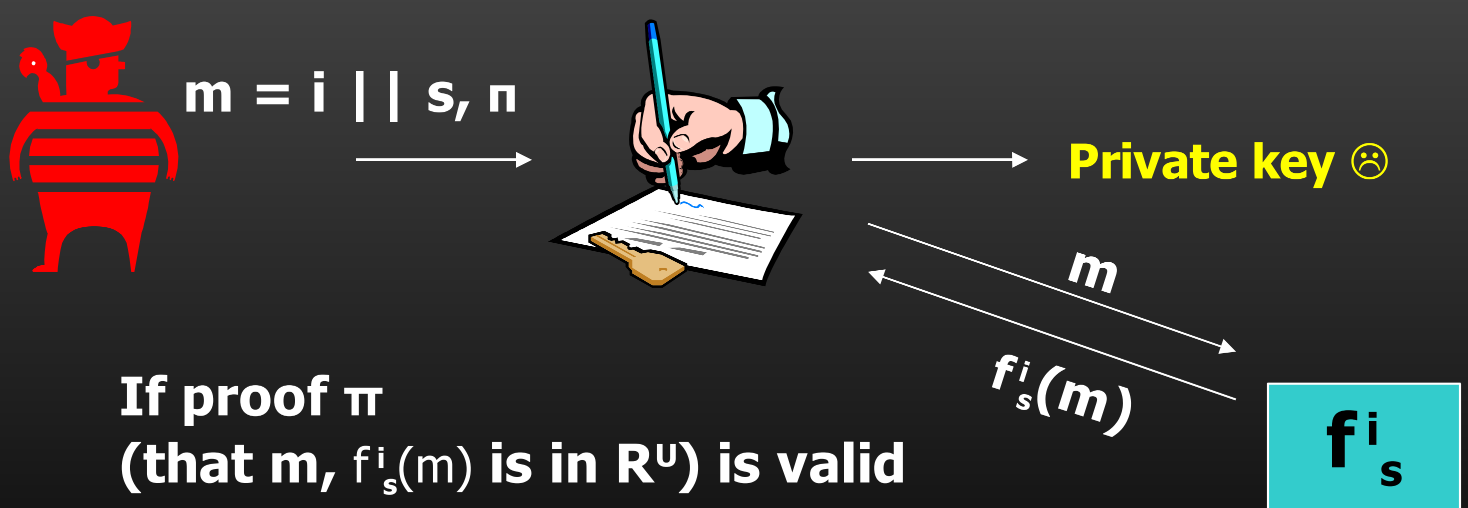
If proof π
(that m, O(m) is in R^U) is valid
give adversary private key
Else sign msg

Scheme using CS Proofs in ROM

- Ideal scheme is secure because
  - R^U is evasive
  - Proof verifier won't accept proofs of untrue statements

Scheme using CS Proofs with Function Ensembles

- Pick an evasive relation $R^U = \{\ i \,||\, x\ ,\ f^i_x(i \,||\, x)\ \}$



If proof π
(that m, $f^i_s(m)$ is in R^U) is valid
give adversary private key
Else sign msg

Scheme using CS Proofs with Function Ensembles

- This scheme is still insecure, because the seed is known to the adversary
  - He can easily satisfy R^U by sending the seed as the message to the Signer
  - It's easy for him to generate a valid proof of the true statement that $(i \| s, f^i_s(i\|s))$ satisfies R^U

Security of CS proofs

- Note that it doesn't matter (to the security of our scheme) whether we can implement CS proofs in the real world
  - The adversary doesn't need to "cheat" the CS proof system
  - He just gives the proof system only valid proofs

So now we're done

What we have shown

- We can build a signature scheme which is secure in ROM, but for which **any** implementation will be insecure.

Does this mean that ROM is useless?

- The schemes we built are contrived
- We had to put in harmful modifications so that the scheme would break in the real world
- Real signature schemes would not be designed this way

We'll talk more about this controversy on Thursday

Questions?