Random Oracles and OAEP

Adam Stubblefield
So far...

- Symmetric encryption
- Two people want to communicate
- Share a secret key
- Want their communication to be private and authenticated
So far...

IND-CPA Symmetric Encryption Scheme + Strongly Unforgable MAC \(\downarrow\)

IND-CCA Authenticated Encryption Scheme
Today

- Symmetric encryption
- Two people want to communicate
- Share a secret key
- Want their communication to be private and authenticated
Today

- **Asymmetric encryption**
- Two people want to communicate
- Don’t share a secret key
- Want their communication to be private and authenticated (?)
Asymmetric Encryption

• Also called *public key encryption*

• Instead of one key that both people share, now there are two per person

• Public key which does not need to be kept secret \((k)\)

• Private key which only the owner should know \((k^{-1})\)
Private key

Public Key

Public Key

Attack at dawn

Private key
Private key

Public Key

Public Key

Encrypt

Private key

Attack at dawn
Private key

Public Key

Public Key

Attack at dawn

Private key
Private key

Public Key

Public Key

Attack at dawn

Decrypt

Private key
Message could have come from anyone

Attack at dawn

Decrypt

Private key

Public Key

Public Key

Private key
A New Atomic Primitive

- Family of one-way trapdoor permutations
- Family of permutations \((f, f^{-1})\)
- One-way means that given \(f\) and \(y\), it’s hard to come up with the \(x\) where \(f(x) = y\)
- The inverse, \(f^{-1}\), is the trapdoor
- Examples: RSA, Rabin, etc...
RSA is a one-way trapdoor permutation, not an encryption scheme.
Just like we built secure symmetric encryption out of PRPs (CTR), we want to build secure asymmetric encryption schemes out of OWTPs (OAEP)

Optimal Asymmetric Encryption Protocol
Message

Attack at dawn
<table>
<thead>
<tr>
<th>Message</th>
<th>Zeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack at dawn</td>
<td>0000000000</td>
</tr>
<tr>
<td>Message</td>
<td>Zeros</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Attack at dawn</td>
<td>0000000000</td>
</tr>
</tbody>
</table>
Message | Zeros | Random bits
---|---|---
Attack at dawn | 0000000000 | 0101101010
Message: Attack at dawn
Zeros: 0000000000
Random bits: 010110101
Message: Attack at dawn
Zeros: 0000000000
Random bits: 0101101010

Diagram:
- Message
- Zeros
- Random bits
- Intermediate nodes G and H
- Output S
Attack at dawn

Zeros: 0000000000

Random bits: 010110101

Message

G

H

s

t
Message: Attack at dawn  
Zeros: 0000000000  
Random bits: 010110101

Public key: $f$  
Private key: $f^{-1}$  

$E(m) = f(s || t)$
\[ s \parallel t = f^{-1}(c) \]
Attack at dawn

0000000000

010110101

G

H

s

t
Message

Attack at dawn

Zeros

0000000000

Random bits

010110101

Diagram:

- msg: Attack at dawn
- zeros: 0000000000
- random_bits: 010110101
- nodes: G, H
- connections:
  - G to H
  - H to G
  - G to t
  - H to t

Output:

s

t
Message | Attack at dawn
---|---
Zeros | 0000000000
Random bits | 010110101

\[ s \]

\[ t \]
The Zeros must all be 0, otherwise we return \( \bot \).
What are G and H?

- Publicly computable (no keys)
- Randomish
- Onewayish
- Collision resistantish
- None of these properties are sufficient
Real Cryptographic Hash Functions

- Unkeyed SHA-1 is (hopefully):
  - Collision resistant
  - One-way
  - “Random looking”
  - And more...
Need Some Way To Model These Functions

- Can’t enumerate all the properties they’re supposed to have, but have some intuition
- We will replace these functions with something that has all the properties that we want hash functions to have, but we’ll overshoot
- No real function has the properties we claim
Random Oracles
Random Oracles

\[ x \rightarrow R \]
Random Oracles

Each bit of the output is chosen uniformly at random.
Random Oracles

\[ y \rightarrow R \rightarrow 110100100111... \]
Random Oracles

On the same input always returns the same output
Random Oracles

If you want a shorter output just ignore the rest
Key Thing To Note

• There’s no way to figure out anything about the output of R when given x short of asking R for the output

• So, if the adversary knows R(x) we know he must have asked R for it
Random Oracles Can’t Exist

- We will *approximate* them with cryptographic hash functions
- We will *prove* that a construction that uses random oracles is secure
- We then implement the construction using cryptographic hash functions and *hope* that the hash functions are a good approximation
Why Does This Make Sense?

• We want to accomplish some real world goal

• Some construction is going to be used no matter what

• If we can’t prove anything about any of the efficient constructions without random oracles, we might as well use one that we can prove secure under the R.O. assumption
Proof of Security

• Similar game to before:
  • Adversary given access to encryption and decryption oracles
  • Also given access to the random oracles $G$ and $H$
  • Given the encryption of either $m_0$ or $m_1$, has to decide which it is
Break OAEP, you’ve broken the OWTP

• Use the adversary that breaks OAEP to break the underlying one-way trapdoor permutation

• If the adversary can win at the $m_0$ or $m_1$ game, we can invert $f$ (i.e. given a $y$, come up with $x$ s.t. $f(x) = y$)
Adversary $B(f, y)$
// Wants to find $x$ s.t. $f(x) = y$
Run $A$
  When $A$ asks for $G(x)$:
    See if $G[x]$ exists, if so return it
    Generate $G[x]$ at random, return it
  When $A$ asks for $H(x)$:
    See if $H[x]$ exists, if so return it
    Generate $H[x]$ at random, return it
...

Adversary $B(f, y)$
// Wants to find $x$ s.t. $f(x) = y$
Run $A$

When $A$ asks for $G(x)$:
  See if $G[x]$ exists, if so return it
  Generate $G[x]$ at random, return it

When $A$ asks for $H(x)$:
  See if $H[x]$ exists, if so return it
  Generate $H[x]$ at random, return it

...
When A asks for $E(m)$:

$$\text{return } f(s \| t)$$
When A asks for $D(c)$:
When A asks for D(c):
When A asks for D(c):
When A asks for D(c):

\[ s = b \]

\[ t = a \oplus H[b] \]
When A asks for D(c):

\[ G[a] \oplus b \]

\[ s = b \]

\[ t = a \oplus H[b] \]
When A asks for $D(c)$:

For index $a$ of $G[]$
For index $b$ of $H[]$
if $f(b || a \oplus H[b]) = c$
if $G[a] \oplus b$ has Zeros
return $G[a] \oplus b$
return $\bot$

$s = b$
$t = a \oplus H[b]$
A gives us $m_0$ and $m_1$

No matter what, we say that
the encryption is $y$
(remember that $y$ is the thing
we’re trying to invert)

What if $y$ isn’t the encryption of
either $m_0$ or $m_1$?
There will be some Random Bits and answers to G and H s.t. 
\[ y = f(s \ || \ t) \]
The only way A can win is if it has asked for G[r] and H[s]

We just look at our tables

\[ y = f(x) = f(s \ || \ t) \]
The Result

- If someone can mount a chosen ciphertext attack on OAEP, they can invert the underlying trapdoor permutation in the random oracle world.
Not So Fast...

- There’s a subtle flaw in the proof
- It took 7 years for someone to find
- OAEP was already being used
- We’ll look at what happened