Chosen Ciphertext Attacks

Chosen Ciphertext Attacks

- The CPA security definition considers only the information leaked to the adversary by honestly-generated ciphertexts.
- However, it does not consider what happens when an adversary is allowed to inject its own maliciously crafted ciphertexts into an honest system.
- If that happens, then even a CPA-secure encryption scheme can fail in spectacular ways.

Padding Oracle Attacks

- Imagine a webserver that receives CBC-encrypted ciphertexts for processing.
- When receiving a ciphertext, the webserver decrypts it under the appropriate key and then checks whether the plaintext has valid X.923 padding (Data is padded with null bytes, except for the last byte of padding which indicates how many padding bytes there are.)

01	34	11	d9	81	88	05	57	1d	73	c3	**00**	**00**	**00**	**00**	**05**	$\Rightarrow$ *valid*
95	51	05	4a	d6	5a	a3	44	af	b3	85	00	00	**00**	**00**	**03**	$\Rightarrow$ *valid*
71	da	77	5a	5e	77	eb	a8	73	c5	50	b5	81	d5	96	**01**	$\Rightarrow$ *valid*
5b	1c	01	41	5d	53	86	4e	e4	94	13	e8	7a	89	c4	**71**	$\Rightarrow$ *invalid*
d4	0d	d8	7b	53	24	c6	d1	af	5f	d6	f6	**00**	**c0**	**00**	**04**	$\Rightarrow$ *invalid*

Padding Oracle Attacks

- No matter how the attacker comes by this information, we say that the attacker has access to a padding oracle:

$$\begin{array}{|l|}\hline \underline{\textsc{paddingoracle}(c):} \\ \quad m := \mathsf{Dec}(k, c) \\ \quad \text{return } \textsc{validpad}(m) \\ \hline \end{array}$$

- We call this a padding oracle because it answers only one specific kind of question about the input. In this case, the answer that it gives is always a single boolean value.
- We can show that an attacker who doesn't know the encryption key k can use a padding oracle alone to decrypt any ciphertext of its choice!

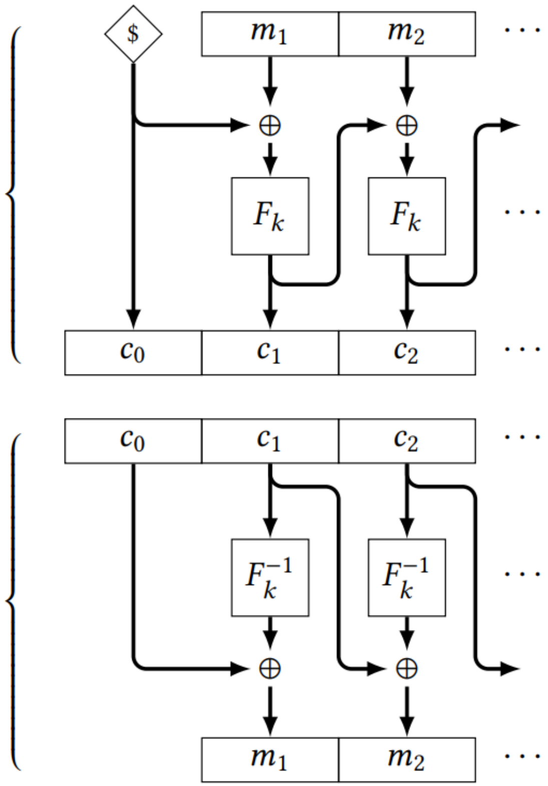
Malleability of CBC Encryption

- Recall the definition of CBC decryption. If the ciphertext is $c = c_0 \cdots c_\ell$ then the ith plaintext block is computed as:

$$m_i := F^{-1}(k, c_i) \oplus c_{i-1}$$

$\underline{\text{Enc}(k, m_1 \| \cdots \| m_\ell):}$
$c_0 \leftarrow \{0,1\}^{blen}$;
for $i = 1$ to ℓ:
  $c_i := F(k, m_i \oplus c_{i-1})$
return $c_0 \| c_1 \| \cdots \| c_\ell$

$\underline{\text{Dec}(k, c_0 \| \cdots \| c_\ell):}$
for $i = 1$ to ℓ:
  $m_i := F^{-1}(k, c_i) \oplus c_{i-1}$
return $m_1 \| \cdots \| m_\ell$



Malleability of CBC Encryption

- Recall the definition of CBC decryption. If the ciphertext is $c = c_0 \cdots c_\ell$ then the ith plaintext block is computed as:
$$m_i := F^{-1}(k, c_i) \oplus c_{i-1}$$
- Two consecutive blocks (c_{i-1}, c_i) taken in isolation are a valid encryption of m_i. This fact allows the attacker to focus on decrypting a single block at a time.
- Xoring a ciphertext block with a known value (say, x) has the effect of xoring the corresponding plaintext block by the same value. In other words, for all x, the ciphertext $(c_{i-1} \oplus x, c_i)$ decrypts to $m_i \oplus x$:
$$\mathrm{Dec}(k, (c_{i-1} \oplus x, c_i)) = F^{-1}(k, c_i) \oplus (c_{i-1} \oplus x) = (F^{-1}(k, c_i) \oplus c_{i-1}) \oplus x = m_i \oplus x$$
- If we send such a ciphertext $(c_{i-1} \oplus x, c_i)$ to the padding oracle, we would therefore learn whether $m_i \oplus x$ is a (single block) with valid padding.
    - Instead of thinking in terms of padding, it might be best to think of the oracle as telling you whether $m_i \oplus x$ ends in one of the suffixes 01 , 00 02 , 00 00 03 , etc.

Malleability of CBC Encryption

- By carefully choosing different values x and asking questions of this form to the padding oracle, we will show how it is possible to learn all of m_i.

```
// suppose c encrypts an (unknown) plaintext m_1 || ··· || m_ℓ
// does m_i ⊕ x end in one of the valid pading strings?

CHECKXOR(c, i, x):
  return PADDINGORACLE(c_{i-1} ⊕ x, c_i)
```

- Given a ciphertext c that encrypts an unknown message m, we can see that an adversary can generate another ciphertext whose contents are related to m in a predictable way.
- This property of an encryption scheme is called malleability.

Learning the Last Byte of a Block

- How to use CHECKXOR to determine the last byte of a plaintext block m.
- Case 1: second-to-last byte of m is nonzero.
  - Try every possible byte b and ask whether $m \oplus b$ has valid padding.
  - Only $m \oplus b$ ends in byte 01 has valid padding.
  - Therefore, if b is the candidate byte that succeeds (i.e., $m \oplus b$ has valid padding) then the last byte of m must be $b \oplus$ 01.

Using LEARNLASTBYTE *to learn the last byte of a plaintext block:*

	$\cdots$	a0	42	??	m = *unknown plaintext block*
$\oplus$	$\cdots$	00	00	b	b = *byte that causes oracle to return* `true`
=	$\cdots$	a0	42	**01**	*valid padding* $\Leftrightarrow b \oplus$?? = 01
					$\Leftrightarrow$?? = 01 $\oplus$ b

Learning the Last Byte of a Block

- How to use CHECKXOR to determine the last byte of a plaintext block m.
- Case 2: second-to-last byte of m is zero. Then $m \oplus b$ will have valid padding for several candidate values of b:

Using LEARNLASTBYTE *to learn the last byte of a plaintext block:*

⋯ a0 00 ??	⋯ a0 00 ??	*m = unknown plaintext*
$\oplus$ ⋯ 00 00 b_1	$\oplus$ ⋯ 00 00 b_2	*b_i = candidate bytes*
= ⋯ a0 00 **01**	= ⋯ a0 **00** **02**	*two candidates cause oracle to return* `true`
↓	↓	
⋯ a0 00 ??	⋯ a0 00 ??	
$\oplus$ ⋯ 00 **01** b_1	$\oplus$ ⋯ 00 **01** b_2	*same b_1, b_2, but change next-to-last byte*
= ⋯ a0 01 **01**	= ⋯ a0 **01** **02**	*only one causes oracle to return* `true` $\Rightarrow$?? $= b_1 \oplus$ 01

Learning the Last Byte of a Block

- How to use CHECKXOR to determine the last byte of a plaintext block m.
- Case 2: second-to-last byte of m is zero. Then $m \oplus b$ will have valid padding for several candidate values of b:
  - Whenever more than one candidate b value yields valid padding, we know that the second-to-last byte of m is zero (in fact, by counting the number of successful candidates, we can know exactly how many zeroes precede the last byte of m).
  - If the second-to-last byte of m is zero, then the second-to-last byte of $m \oplus$ 01 b is nonzero.
  - The only way for both strings $m \oplus$ 01 b and $m \oplus b$ to have valid padding is when $m \oplus b$ ends in byte 01.

Learning Other Bytes of a Block

- Suppose we know the last 3 bytes of a plaintext block. We would like to use the padding oracle to discover the 4th-to-last byte.
- In the worst case, this subroutine makes 256 queries to the padding oracle.

Using LEARNPREVBYTE *to learn the 4th-to-last byte when the last 3 bytes of the block are already known.*

	$\cdots$	??	**a0**	**42**	**3c**	m = *partially unknown plaintext block*
$\oplus$	$\cdots$	00	00	00	04	p = *string ending in* 04
$\oplus$	$\cdots$	00	a0	42	3c	s = *known bytes of* m
$\oplus$	$\cdots$	b	00	00	00	y = *candidate byte* b *shifted into place*
$=$	$\cdots$	00	00	00	04	*valid padding* $\Leftrightarrow$?? $=$ b

What Went Wrong?

- CBC encryption (in fact, every encryption scheme we've seen so far) has a property called malleability. Given an encryption c of an unknown plaintext m, it is possible to generate another ciphertext c' whose contents are related to m in a predictable way.
  - In the case of CBC encryption, if ciphertext $c_0 \| \cdots \| c_\ell$ encrypts a plaintext $m_1 \| \cdots \| m_\ell$, then ciphertext $(c_{i-1} \oplus x, c_i)$ encrypts the related plaintext $m_i \oplus x$.
- Decryption has no impact on CPA security! But the padding oracle setting involved the Dec algorithm.
- The attack makes 256 queries per byte of plaintext, so it costs about 256ℓ queries for a plaintext of ℓ bytes. Brute-forcing the entire plaintext would cost 256^ℓ since that's how many ℓ-byte plaintexts there are. So the attack is exponentially better than brute force.

Defining CCA Security

- How can we possibly anticipate every kind of partial information that might make its way to the adversary in every possible usage of the encryption scheme?
- Let’s just allow the adversary to totally decrypt arbitrary ciphertexts of its choice.
- Simply providing unrestricted Dec access to the adversary cannot lead to a reasonable security definition.
- Allow the adversary to ask for the decryption of any ciphertext, except those produced in response to eavesdrop queries.

Defining CCA Security

Definition Let Σ be an encryption scheme. We say that Σ has security against chosen-ciphertext attacks (CCA security) if $\mathcal{L}^{\Sigma}_{\text{cca-L}} \approx \mathcal{L}^{\Sigma}_{\text{cca-R}}$, where:

$\mathcal{L}^{\Sigma}_{\text{cca-L}}$

$k \leftarrow \Sigma.\text{KeyGen}$
$\mathcal{S} := \emptyset$
 	
$\underline{\text{EAVESDROP}(m_L, m_R \in \Sigma.\mathcal{M})\text{:}}$
  if $\lvert m_L\rvert \neq \lvert m_R\rvert$ return err
  $c := \Sigma.\text{Enc}(k, m_L)$
  $\mathcal{S} := \mathcal{S} \cup \{c\}$
  return c
 	
$\underline{\text{DECRYPT}(c \in \Sigma.\mathcal{C})\text{:}}$
  if $c \in \mathcal{S}$ return err
  return $\Sigma.\text{Dec}(k, c)$

$\mathcal{L}^{\Sigma}_{\text{cca-R}}$

$k \leftarrow \Sigma.\text{KeyGen}$
$\mathcal{S} := \emptyset$
 	
$\underline{\text{EAVESDROP}(m_L, m_R \in \Sigma.\mathcal{M})\text{:}}$
  if $\lvert m_L\rvert \neq \lvert m_R\rvert$ return err
  $c := \Sigma.\text{Enc}(k, m_R)$
  $\mathcal{S} := \mathcal{S} \cup \{c\}$
  return c
 	
$\underline{\text{DECRYPT}(c \in \Sigma.\mathcal{C})\text{:}}$
  if $c \in \mathcal{S}$ return err
  return $\Sigma.\text{Dec}(k, c)$

An Example

- Consider the adversary below attacking the CCA security of CBC mode (with block length $blen$)

$$\begin{array}{|l|}\hline \multicolumn{1}{|c|}{\mathcal{A}} \\ \hline c = c_0 \| c_1 \| c_2 := \text{EAVESDROP}(\mathtt{0}^{2blen}, \mathtt{1}^{2blen}) \\ m := \text{DECRYPT}(c_0 \| c_1) \\ \text{return } m \stackrel{?}{=} \mathtt{0}^{blen} \\ \hline \end{array}$$

- If $c_0 \| c_1 \| c_2$ encrypts $m_1 \| m_2$, then $c_0 \| c_1$ encrypts m_1.

Pseudorandom Ciphertexts

- **Definition** Let Σ be an encryption scheme. We say that Σ has pseudorandom ciphertexts in the presence of chosen-ciphertext attacks (CCA$ security) if $\mathcal{L}^{\Sigma}_{\text{cca\$-real}} \approx \mathcal{L}^{\Sigma}_{\text{cca\$-rand}}$, where:

Just like for CPA security, if a scheme has CCA$ security, then it also has CCA security, but not vice-versa.

$\mathcal{L}^{\Sigma}_{\text{cca\$-real}}$	$\mathcal{L}^{\Sigma}_{\text{cca\$-rand}}$
$k \leftarrow \Sigma.\text{KeyGen}$ $\mathcal{S} := \emptyset$ $\underline{\textsc{ctxt}(m \in \Sigma.\mathcal{M}):}$ $\quad c := \Sigma.\text{Enc}(k, m)$ $\quad \mathcal{S} := \mathcal{S} \cup \{c\}$ $\quad$ return c $\underline{\textsc{decrypt}(c \in \Sigma.\mathcal{C}):}$ $\quad$ if $c \in \mathcal{S}$ return err $\quad$ return $\Sigma.\text{Dec}(k, c)$	$k \leftarrow \Sigma.\text{KeyGen}$ $\mathcal{S} := \emptyset$ $\underline{\textsc{ctxt}(m \in \Sigma.\mathcal{M}):}$ $\quad c \leftarrow \Sigma.\mathcal{C}(\|m\|)$ $\quad \mathcal{S} := \mathcal{S} \cup \{c\}$ $\quad$ return c $\underline{\textsc{decrypt}(c \in \Sigma.\mathcal{C}):}$ $\quad$ if $c \in \mathcal{S}$ return err $\quad$ return $\Sigma.\text{Dec}(k, c)$

A Simple CCA-Secure Scheme

- Let F be a strong pseudorandom permutation with block length $blen = n + \lambda$. Define the following encryption scheme with message space $\mathcal{M} = \{0,1\}^n$:

KeyGen:	Enc(k, m):	Dec(k, c):
$k \leftarrow \{0,1\}^\lambda$	$r \leftarrow \{0,1\}^\lambda$	$v := F^{-1}(k, c)$
return k	return $F(k, m \| r)$	return first n bits of v

- We can informally reason about the security of this scheme as follows:
  - As long as the random value r does not repeat, all inputs to the PRP are distinct, and thus its outputs will therefore all look independently uniform.
  - For any other value c' that the adversary asks to be decrypted, the guarantee of a strong PRP is that the result will look independently random. In particular, the result will not depend on the choice of plaintexts used to generate challenge ciphertexts.

Advanced Cryptography

（Provable Security）

Yi LIU

A Simple CCA-Secure Scheme

KeyGen:	Enc(k, m):	Dec(k, c):
$k \leftarrow \{0,1\}^\lambda$	$r \leftarrow \{0,1\}^\lambda$	$v := F^{-1}(k, c)$
return k	return $F(k, m \| r)$	return first n bits of v

Claim If F is a strong PRP then the construction has CCA$ security (and therefore CCA security).

A Simple CCA-Secure Scheme

Claim If F is a strong PRP then the construction has CCA\$ security (and therefore CCA security).

proof

$\mathcal{L}^{\Sigma}_{\text{cca\$-real}}$

$k \leftarrow \{0,1\}^\lambda$
$\mathcal{S} := \emptyset$

$\textsc{ctxt}(m)$:
  $r \leftarrow \{0,1\}^\lambda$
  $c := F(k, m\|r)$
  $\mathcal{S} := \mathcal{S} \cup \{c\}$
  return c

$\textsc{decrypt}(c \in \Sigma.\mathcal{C})$:
  if $c \in \mathcal{S}$ return err
  return first n bits of $F^{-1}(k, c)$

strong PRP security

$\mathcal{S} := \emptyset$
T, T_{inv} := empty assoc. arrays

$\textsc{ctxt}(m)$:
  $r \leftarrow \{0,1\}^\lambda$
  if $T[m\|r]$ undefined:
    $c \leftarrow \{0,1\}^{blen} \setminus T.\text{values}$
    $T[m\|r] := c$; $T_{inv}[c] := m\|r$
  $c := T[m\|r]$
  $\mathcal{S} := \mathcal{S} \cup \{c\}$
  return c

$\textsc{decrypt}(c \in \Sigma.\mathcal{C})$:
  if $c \in \mathcal{S}$ return err
  if $T_{inv}[c]$ undefined:
    $m\|r \leftarrow \{0,1\}^{blen} \setminus T_{inv}.\text{values}$
    $T_{inv}[c] := m\|r$; $T[m\|r] := c$
  return first n bits of $T_{inv}[c]$

A Simple CCA-Secure Scheme

Claim If F is a strong PRP then the construction has CCA$ security (and therefore CCA security).

proof

$\mathcal{S} := \emptyset$
$T, T_{inv} :=$ empty assoc. arrays

<u>CTXT(m):</u>
  $r \leftarrow \{0,1\}^\lambda$
  if $T[m\|r]$ undefined:
    $c \leftarrow \{0,1\}^{blen} \setminus T$.values
    $T[m\|r] := c$; $T_{inv}[c] := m\|r$
  $c := T[m\|r]$
  $\mathcal{S} := \mathcal{S} \cup \{c\}$
  return c

<u>DECRYPT($c \in \Sigma.\mathcal{C}$):</u>
  if $c \in \mathcal{S}$ return err
  if $T_{inv}[c]$ undefined:
    $m\|r \leftarrow \{0,1\}^{blen} \setminus T_{inv}$.values
    $T_{inv}[c] := m\|r$; $T[m\|r] := c$
  return first n bits of $T_{inv}[c]$

To prove CCA$-security, we must reach a hybrid in which the responses of CTXT are uniform.
In the current hybrid there are two properties in the way of this goal:

- The ciphertext values c are sampled from $\{0,1\}^{blen} \backslash \mathrm{T.values}$, rather than $\{0,1\}^{blen}$.
- To show CCA$ security, we must remove the dependence of DECRYPT on previous values given to CTXT.

A Simple CCA-Secure Scheme

Claim If F is a strong PRP then the construction has CCA$ security (and therefore CCA security).

proof

$\mathcal{S} := \emptyset$
$T, T_{inv} :=$ empty assoc. arrays

$\underline{\text{CTXT}(m):}$
  $r \leftarrow \{0,1\}^\lambda$
  if $T[m\|r]$ undefined:
    $c \leftarrow \{0,1\}^{blen} \setminus T.\text{values}$
    $T[m\|r] := c$; $T_{inv}[c] := m\|r$
  $c := T[m\|r]$
  $\mathcal{S} := \mathcal{S} \cup \{c\}$
  return c

$\underline{\text{DECRYPT}(c \in \Sigma.\mathcal{C}):}$
  if $c \in \mathcal{S}$ return err
  if $T_{inv}[c]$ undefined:
    $m\|r \leftarrow \{0,1\}^{blen} \setminus T_{inv}.\text{values}$
    $T_{inv}[c] := m\|r$; $T[m\|r] := c$
  return first n bits of $T_{inv}[c]$

Add some book-keeping that is not used anywhere.

$\mathcal{S} := \emptyset$; $\mathcal{R} := \emptyset$
$T, T_{inv} :=$ empty assoc. arrays

$\underline{\text{CTXT}(m):}$
  $r \leftarrow \{0,1\}^\lambda$
  if $T[m\|r]$ undefined:
    $c \leftarrow \{0,1\}^{blen} \setminus T.\text{values}$
    $T[m\|r] := c$; $T_{inv}[c] := m\|r$
    $\mathcal{R} := \mathcal{R} \cup \{r\}$
  $c := T[m\|r]$
  $\mathcal{S} := \mathcal{S} \cup \{c\}$
  return c

$\underline{\text{DECRYPT}(c \in \Sigma.\mathcal{C}):}$
  if $c \in \mathcal{S}$ return err
  if $T_{inv}[c]$ undefined:
    $m\|r \leftarrow \{0,1\}^{blen} \setminus T_{inv}.\text{values}$
    $T_{inv}[c] := m\|r$; $T[m\|r] := c$
    $\mathcal{R} := \mathcal{R} \cup \{r\}$
  return first n bits of $T_{inv}[c]$

A Simple CCA-Secure Scheme

Claim If F is a strong PRP then the construction has CCA\$ security (and therefore CCA security).

proof

$\mathcal{S} := \emptyset$; $\mathcal{R} := \emptyset$
T, T_{inv} := empty assoc. arrays

$\underline{\text{CTXT}(m)\text{:}}$
  $r \leftarrow \{0,1\}^\lambda$
  if $T[m\|r]$ undefined:
    $c \leftarrow \{0,1\}^{blen} \setminus T.\text{values}$
    $T[m\|r] := c$; $T_{inv}[c] := m\|r$
    $\mathcal{R} := \mathcal{R} \cup \{r\}$
  $c := T[m\|r]$
  $\mathcal{S} := \mathcal{S} \cup \{c\}$
  return c

$\underline{\text{DECRYPT}(c \in \Sigma.\mathcal{C})\text{:}}$
  if $c \in \mathcal{S}$ return err
  if $T_{inv}[c]$ undefined:
    $m\|r \leftarrow \{0,1\}^{blen} \setminus T_{inv}.\text{values}$
    $T_{inv}[c] := m\|r$; $T[m\|r] := c$
    $\mathcal{R} := \mathcal{R} \cup \{r\}$
  return first n bits of $T_{inv}[c]$

Apply replacement vs without replacement three separate times.

$\Rightarrow$

$\mathcal{S} := \emptyset$; $\mathcal{R} := \emptyset$
T, T_{inv} := empty assoc. arrays

$\underline{\text{CTXT}(m)\text{:}}$
  $r \leftarrow \{0,1\}^\lambda \setminus \mathcal{R}$
  if $T[m\|r]$ undefined:
    $c \leftarrow \{0,1\}^{blen}$
    $T[m\|r] := c$; $T_{inv}[c] := m\|r$
    $\mathcal{R} := \mathcal{R} \cup \{r\}$
  $c := T[m\|r]$
  $\mathcal{S} := \mathcal{S} \cup \{c\}$
  return c

$\underline{\text{DECRYPT}(c \in \Sigma.\mathcal{C})\text{:}}$
  if $c \in \mathcal{S}$ return err
  if $T_{inv}[c]$ undefined:
    $m\|r \leftarrow \{0,1\}^{blen}$
    $T_{inv}[c] := m\|r$; $T[m\|r] := c$
    $\mathcal{R} := \mathcal{R} \cup \{r\}$
  return first n bits of $T_{inv}[c]$

A Simple CCA-Secure Scheme

Claim If F is a strong PRP then the construction has CCA$ security (and therefore CCA security).

proof

$\mathcal{S} := \emptyset; \quad \mathcal{R} := \emptyset$
$T, T_{inv} :=$ empty assoc. arrays

$\underline{\text{CTXT}(m):}$
$\quad r \leftarrow \{0,1\}^\lambda \setminus \mathcal{R}$
$\quad$ if $T[m\|r]$ undefined:
$\quad\quad c \leftarrow \{0,1\}^{blen}$
$\quad\quad T[m\|r] := c; T_{inv}[c] := m\|r$
$\quad\quad \mathcal{R} := \mathcal{R} \cup \{r\}$
$\quad c := T[m\|r]$
$\quad \mathcal{S} := \mathcal{S} \cup \{c\}$
$\quad$ return c

$\underline{\text{DECRYPT}(c \in \Sigma.\mathcal{C}):}$
$\quad$ if $c \in \mathcal{S}$ return err
$\quad$ if $T_{inv}[c]$ undefined:
$\quad\quad m\|r \leftarrow \{0,1\}^{blen}$
$\quad\quad T_{inv}[c] := m\|r; T[m\|r] := c$
$\quad\quad \mathcal{R} := \mathcal{R} \cup \{r\}$
$\quad$ return first n bits of $T_{inv}[c]$

The if-statement in CTXT is always taken.

➡

$\mathcal{S} := \emptyset; \quad \mathcal{R} := \emptyset$
$T, T_{inv} :=$ empty assoc. arrays

$\underline{\text{CTXT}(m):}$
$\quad r \leftarrow \{0,1\}^\lambda \setminus \mathcal{R}$
$\quad c \leftarrow \{0,1\}^{blen}$
$\quad T[m\|r] := c; T_{inv}[c] := m\|r$
$\quad \mathcal{R} := \mathcal{R} \cup \{r\}$
$\quad \mathcal{S} := \mathcal{S} \cup \{c\}$
$\quad$ return c

$\underline{\text{DECRYPT}(c \in \Sigma.\mathcal{C}):}$
$\quad$ if $c \in \mathcal{S}$ return err
$\quad$ if $T_{inv}[c]$ undefined:
$\quad\quad m\|r \leftarrow \{0,1\}^{blen}$
$\quad\quad T_{inv}[c] := m\|r; T[m\|r] := c$
$\quad\quad \mathcal{R} := \mathcal{R} \cup \{r\}$
$\quad$ return first n bits of $T_{inv}[c]$

A Simple CCA-Secure Scheme

Claim If F is a strong PRP then the construction has CCA$ security (and therefore CCA security).

proof

$\mathcal{S} := \emptyset; \quad \mathcal{R} := \emptyset$
$T, T_{inv} :=$ empty assoc. arrays

$\underline{\text{CTXT}(m):}$
$\quad r \leftarrow \{0,1\}^\lambda \setminus \mathcal{R}$
$\quad c \leftarrow \{0,1\}^{blen}$
$\quad T[m\|r] := c; T_{inv}[c] := m\|r$
$\quad \mathcal{R} := \mathcal{R} \cup \{r\}$
$\quad \mathcal{S} := \mathcal{S} \cup \{c\}$
$\quad$ return c

$\underline{\text{DECRYPT}(c \in \Sigma.\mathcal{C}):}$
$\quad$ if $c \in \mathcal{S}$ return err
$\quad$ if $T_{inv}[c]$ undefined:
$\quad\quad m\|r \leftarrow \{0,1\}^{blen}$
$\quad\quad T_{inv}[c] := m\|r; T[m\|r] := c$
$\quad\quad \mathcal{R} := \mathcal{R} \cup \{r\}$
$\quad$ return first n bits of $T_{inv}[c]$

- No line of code ever reads from T
- The first line of DECRYPT returns err for $c \in \mathcal{S}$. So T_{inv} is not read.

$\Rightarrow$

$\mathcal{S} := \emptyset; \quad \mathcal{R} := \emptyset$
$T, T_{inv} :=$ empty assoc. arrays

$\underline{\text{CTXT}(m):}$
$\quad r \leftarrow \{0,1\}^\lambda \setminus \mathcal{R}$
$\quad c \leftarrow \{0,1\}^{blen}$
$\quad$ // $T[m\|r] := c; T_{inv}[c] := m\|r$
$\quad \mathcal{R} := \mathcal{R} \cup \{r\}$
$\quad \mathcal{S} := \mathcal{S} \cup \{c\}$
$\quad$ return c

$\underline{\text{DECRYPT}(c \in \Sigma.\mathcal{C}):}$
$\quad$ if $c \in \mathcal{S}$ return err
$\quad$ if $T_{inv}[c]$ undefined:
$\quad\quad m\|r \leftarrow \{0,1\}^{blen}$
$\quad\quad T_{inv}[c] := m\|r; T[m\|r] := c$
$\quad\quad \mathcal{R} := \mathcal{R} \cup \{r\}$
$\quad$ return first n bits of $T_{inv}[c]$

A Simple CCA-Secure Scheme

Claim If F is a strong PRP then the construction has CCA$ security (and therefore CCA security).

proof

$\mathcal{S} := \emptyset$; $\mathcal{R} := \emptyset$
T, T_{inv} := empty assoc. arrays

<u>CTXT(m):</u>
$r \leftarrow \{0,1\}^\lambda \setminus \mathcal{R}$
$c \leftarrow \{0,1\}^{blen}$
// $T[m\|r] := c$; $T_{inv}[c] := m\|r$
$\mathcal{R} := \mathcal{R} \cup \{r\}$
$\mathcal{S} := \mathcal{S} \cup \{c\}$
return c

<u>DECRYPT($c \in \Sigma.\mathcal{C}$):</u>
if $c \in \mathcal{S}$ return err
if $T_{inv}[c]$ undefined:
$m\|r \leftarrow \{0,1\}^{blen}$
$T_{inv}[c] := m\|r$; $T[m\|r] := c$
$\mathcal{R} := \mathcal{R} \cup \{r\}$
return first n bits of $T_{inv}[c]$

It has no effect to simply remove all lines that refer to variable $\mathcal{R}$.

$\mathcal{S} := \emptyset$; // $\mathcal{R} := \emptyset$
T, T_{inv} := empty assoc. arrays

<u>CTXT(m):</u>
// $r \leftarrow \{0,1\}^\lambda \setminus \mathcal{R}$
$c \leftarrow \{0,1\}^{blen}$
// $\mathcal{R} := \mathcal{R} \cup \{r\}$
$\mathcal{S} := \mathcal{S} \cup \{c\}$
return c

<u>DECRYPT($c \in \Sigma.\mathcal{C}$):</u>
if $c \in \mathcal{S}$ return err
if $T_{inv}[c]$ undefined:
$m\|r \leftarrow \{0,1\}^{blen}$
$T_{inv}[c] := m\|r$; $T[m\|r] := c$
// $\mathcal{R} := \mathcal{R} \cup \{r\}$
return first n bits of $T_{inv}[c]$

A Simple CCA-Secure Scheme

Claim If F is a strong PRP then the construction has CCA\$ security (and therefore CCA security).

proof

$\mathcal{S} := \emptyset$; // $\mathcal{R} := \emptyset$
$T, T_{inv} :=$ empty assoc. arrays

<u>CTXT(m):</u>
  // $r \leftarrow \{0,1\}^{\lambda} \setminus \mathcal{R}$
  $c \leftarrow \{0,1\}^{blen}$
  // $\mathcal{R} := \mathcal{R} \cup \{r\}$
  $\mathcal{S} := \mathcal{S} \cup \{c\}$
  return c

<u>DECRYPT($c \in \Sigma.\mathcal{C}$):</u>
  if $c \in \mathcal{S}$ return err
  if $T_{inv}[c]$ undefined:
    $m \| r \leftarrow \{0,1\}^{blen}$
    $T_{inv}[c] := m\|r$; $T[m\|r] := c$
    // $\mathcal{R} := \mathcal{R} \cup \{r\}$
  return first n bits of $T_{inv}[c]$

$\Rightarrow$

$\mathcal{S} := \emptyset$
$T, T_{inv} :=$ empty assoc. arrays

<u>CTXT(m):</u>
  $c \leftarrow \{0,1\}^{blen}$
  $\mathcal{S} := \mathcal{S} \cup \{c\}$
  return c

<u>DECRYPT($c \in \Sigma.\mathcal{C}$):</u>
  if $c \in \mathcal{S}$ return err
  if $T_{inv}[c]$ undefined:
    $m \| r \leftarrow \{0,1\}^{blen} \setminus T_{inv}.\text{values}$
    $T_{inv}[c] := m\|r$; $T[m\|r] := c$
  return first n bits of $T_{inv}[c]$

A Simple CCA-Secure Scheme

Claim If F is a strong PRP then the construction has CCA$ security (and therefore CCA security).

proof

$\mathcal{S} := \emptyset$
$T, T_{inv} :=$ empty assoc. arrays

$\underline{\textsc{ctxt}(m):}$
  $c \leftarrow \{0,1\}^{blen}$
  $\mathcal{S} := \mathcal{S} \cup \{c\}$
  return c

$\underline{\textsc{decrypt}(c \in \Sigma.\mathcal{C}):}$
  if $c \in \mathcal{S}$ return err
  if $T_{inv}[c]$ undefined:
    $m \| r \leftarrow \{0,1\}^{blen} \setminus T_{inv}.\text{values}$
    $T_{inv}[c] := m\|r$; $T[m\|r] := c$
  return first n bits of $T_{inv}[c]$

$\mathcal{L}^{\Sigma}_{\text{cca\$-rand}}$

$k \leftarrow \{0,1\}^{\lambda}$
$\mathcal{S} := \emptyset$

$\underline{\textsc{ctxt}(m):}$
  $c \leftarrow \{0,1\}^{blen}$
  $\mathcal{S} := \mathcal{S} \cup \{c\}$
  return c

$\underline{\textsc{decrypt}(c \in \Sigma.\mathcal{C}):}$
  if $c \in \mathcal{S}$ return err
  return first n bits of $F^{-1}(k, c)$

One-Way Function

One-Way Function

- A one-way function $f: \{0,1\}^* \to \{0,1\}^*$ is easy to compute, yet hard to invert.
- **Easy to compute**: f is computable in polynomial time
- **Hard to invert**: It is infeasible for any probabilistic polynomial-time algorithm to invert f—that is, to find a preimage of a given value y—except with negligible probability.

One-Way Function

Let $f: \{0,1\}^* \rightarrow \{0,1\}^*$ be a function. Consider the following experiment defined for any algorithm $\mathcal{A}$ and any value λ for the security parameter:

The inverting experiment $\mathbf{Invert}_{\mathcal{A},f}(\lambda)$

1. Choose uniform $x \in \{0,1\}^\lambda$, and compute $y := f(x)$.
2. The algorithm $\mathcal{A}$ is given 1^λ and y as input, and outputs x'.
3. The output of the experiment is defined to be 1 if $f(x') = y$, and 0 otherwise.

- We stress that $\mathcal{A}$ need not find the original preimage x; it suffices for $\mathcal{A}$ to find any value x' for which $f(x') = y = f(x)$.
- The security parameter 1^λ is given to $\mathcal{A}$ in the second step to stress that $\mathcal{A}$ may run in time polynomial in the security parameter λ, regardless of the length of y.

One-Way Function

A function $f: \{0,1\}^* \rightarrow \{0,1\}^*$ is one-way if the following two conditions hold:

1. (**Easy to compute**:) There exists a polynomial-time algorithm M_f computing f; that is, $M_f(x) = f(x)$ for all x.
2. (**Hard to invert**:) For every probabilistic polynomial-time algorithm $\mathcal{A}$, there is a negligible function negl such that
$$\Pr[\mathbf{Invert}_{\mathcal{A},f}(\lambda) = 1] \leq \mathrm{negl}(\lambda)$$

$\Pr[\mathbf{Invert}_{\mathcal{A},f}(\lambda) = 1] \leq \mathrm{negl}(\lambda)$ can be rewritten as

$$\Pr_{x \leftarrow \{0,1\}^\lambda}\left[\mathcal{A}\left(1^\lambda, f(x)\right) \in f^{-1}\big(f(x)\big)\right] \leq \mathrm{negl}(\lambda)$$

Candidate One-Way Functions

- $f(p,q) = p \times q$ for large equal-length primes p and q.
- subset-sum problem
  - $f_{ss}(x_1, \ldots, x_n, J) = \left(x_1, \ldots, x_n, \left[\sum_{j \in J} x_j \bmod 2^n\right]\right)$
  - where each x_i is an n-bit string interpreted as an integer, and J is an n-bit string interpreted as specifying a subset of $\{1, \ldots, n\}$
- $f_{p,g}(x) = [g^x \bmod p]$
  - p is an λ-bit prime. g is the generator of $\mathbb{Z}_p^*$
- SHA-2 or AES under the assumption that they are collision resistant or a pseudorandom permutation, respectively.

Hard-Core Predicates

- One-way function: given $y = f(x)$, the value x cannot be computed in its entirety by any polynomial-time algorithm (except with negligible probability)
- Does this mean that nothing about x can be determined from $f(x)$ in polynomial time?
- **No**. It is possible for $f(x)$ to "leak" a lot of information about x even if f is one-way.
  - let g be a one-way function and define $f(x_1, x_2) = (x_1, g(x_2))$, where $|x_1| = |x_2|$.
  - It is easy to show that f is also a one-way function, even though it reveals half its input.
- For our applications, we will need to identify a specific piece of information about x that is "hidden" by $f(x)$. This motivates the notion of a *hard-core predicate*.

Hard-Core Predicates

A hard-core predicate $\mathrm{hc} : \{0,1\}^* \to \{0,1\}$ of a function f has the property that $\mathrm{hc}(x)$ is hard to compute with probability significantly better than 1/2 given $f(x)$. (Since hc is a boolean function, it is always possible to compute $\mathrm{hc}(x)$ with probability 1/2 by random guessing.)

Definition A function $\mathrm{hc} : \{0,1\}^* \to \{0,1\}$ is a hard-core predicate of a function f if hc can be computed in polynomial time, and for every probabilistic polynomial-time algorithm $\mathcal{A}$ there is a negligible function negl such that

$$\Pr_{x \leftarrow \{0,1\}^\lambda}[A(1^\lambda, f(x)) = \mathrm{hc}(x)] \leq 1/2 + \mathrm{negl}(\lambda)$$

where the probability is taken over the uniform choice of x in $\{0,1\}^\lambda$ and the randomness of $\mathcal{A}$.

We stress that $\mathrm{hc}(x)$ is efficiently computable given x (since the function hc can be computed in polynomial time); the definition requires that $\mathrm{hc}(x)$ is hard to compute given $f(x)$.

Simple ideas don’t work

Consider the predicate $\mathrm{hc}(x) = \bigoplus_{i=1}^{\lambda} x_i$ where $x_1, \ldots, x_\lambda$ denote the bits of x. Is this a hard-core predicate of any one-way function f?

- If f cannot be inverted, then $f(x)$ must hide at least one of the bits x_i of its preimage x, which would seem to imply that the XOR of all of the bits of x is hard to compute.
- This argument is **incorrect**
  - Let g be a one-way function and define $f(x) = (g(x), \bigoplus_{i=1}^{\lambda} x_i)$. It is not hard to show that f is one-way. However, it is clear that $f(x)$ does not hide the value of $\mathrm{hc}(x) = \bigoplus_{i=1}^{\lambda} x_i$ because this is part of its output. Therefore, $\mathrm{hc}(x)$ is not a hard-core predicate of f.
  - For any fixed predicate hc, there is a one-way function f for which hc is not a hard-core predicate of f.

Trivial hard-core predicates

- Some functions have "trivial" hard-core predicates.
- Let f be the function that drops the last bit of its input (i.e., $f(x_1 \cdots x_\lambda) = x_1 \cdots x_{\lambda-1}$). It is hard to determine x_λ given $f(x)$ since x_λ is independent of the output; thus, $\mathrm{hc}(x) = x_\lambda$ is a hard-core predicate of f.
- However, f is not one-way.
- Trivial hard-core predicates of this sort are of no use.

Hard-Core Predicate from One-Way Functions

Theorem (Goldreich–Levin theorem) Assume one-way functions (resp., permutations) exist. Then there exists a one-way function (resp., permutation) g and a hard-core predicate gl of g.

Let f be a one-way function. Functions g and gl are constructed as follows: set $g(x, r) = (f(x), r)$, for $|x| = |r|$, and define

$$\mathrm{gl}(x, r) = \bigoplus_{i=1}^{\lambda} x_i \cdot r_i$$

where x_i (resp., r_i) denotes the ith bit of x (resp., r).

Notice that if r is uniform, then $\mathrm{gl}(x, r)$ outputs the XOR of a random subset of the bits of x. (When $r_i = 1$ the bit x_i is included in the XOR, and otherwise it is not.) The Goldreich–Levin theorem thus states that if f is a one-way function then $f(x)$ hides the XOR of a random subset of the bits of x.

Hard-Core Predicate from One-Way Functions

Theorem Let f be a one-way function and define $g(x, r) = (f(x), r)$, where $|x| = |r|$, and $\mathrm{gl}(x, r) = \bigoplus_{i=1}^{\lambda} x_i \cdot r_i$. Then gl is a hard-core predicate of g.

We first show that if there exists a polynomial-time adversary $\mathcal{A}$ that always correctly computes $\mathrm{gl}(x, r)$ given $g(x, r) = (f(x), r)$, then it is possible to invert f in polynomial time.

Hard-Core Predicate from One-Way Functions

Proposition Let f and gl be as before. If there exists a polynomial-time algorithm $\mathcal{A}$ such that $\mathcal{A}(f(x), r) = \mathrm{gl}(x, r)$ for all λ and all $x, r \in \{0,1\}^\lambda$, then there exists a polynomial-time algorithm $\mathcal{A}'$ such that $\mathcal{A}'(1^\lambda, f(x)) = x$ for all n and all $x \in \{0,1\}^\lambda$.

If f is one-way, it is impossible for any probabilistic polynomial-time algorithm to invert f with non-negligible probability. Thus, we conclude that there is no polynomial-time algorithm that always correctly computes $\mathrm{gl}(x, r)$ from $(f(x), r)$.

This is a rather weak result that is very far from our ultimate goal of showing that $\mathrm{gl}(x, r)$ cannot be computed with probability significantly better than $1/2$ given $(f(x), r)$.

A Simple Case

Proposition Let f and gl be as before. If there exists a polynomial-time algorithm $\mathcal{A}$ such that $\mathcal{A}(f(x), r) = \mathrm{gl}(x, r)$ for all n and all $x, r \in \{0,1\}^\lambda$, then there exists a polynomial-time algorithm $\mathcal{A}'$ such that $\mathcal{A}'(1^\lambda, f(x)) = x$ for all n and all $x \in \{0,1\}^\lambda$.

proof

$\mathcal{A}'(1^\lambda, y)$ computes $x_i = \mathcal{A}(y, e^i)$ for $i = 1, \ldots, \lambda$, where e^i denotes the λ-bit string with 1 in the ith position and 0 everywhere else. Then $\mathcal{A}'$ outputs $x = x_1 \cdots x_\lambda$. Clearly, $\mathcal{A}'$ runs in polynomial time.

In the execution of $\mathcal{A}'\left(1^\lambda, f(\hat{x})\right)$,

$$x_i = \mathcal{A}\left(f(\hat{x}), e^i\right) = \mathrm{gl}\left(\hat{x}, e^i\right) = \bigoplus_{j=1}^{\lambda} \hat{x}_j \cdot e_j^i = \hat{x}_i$$

Thus $x_i = \hat{x}_i$ for all i. The output of $\mathcal{A}$ is the correct inverse.

A More Involved Case

- We now show that it is hard for any probabilistic polynomial-time algorithm $\mathcal{A}$ to compute $\mathrm{gl}(x, r)$ from $(f(x), r)$ with probability significantly better than $3/4$.
- We will again show that any such $\mathcal{A}$ would imply the existence of a polynomial-time algorithm $\mathcal{A}'$ that inverts f with non-negligible probability.
- For the simple case strategy:
  - It may be that $\mathcal{A}$ never succeeds when $r = e^i$ (although it may succeed, say, on all other values of r).
  - $\mathcal{A}'$ does not know if the result $\mathcal{A}(f(x), r)$ is equal to $\mathrm{gl}(x, r)$ or not.

A More Involved Case

Proposition Let f and gl be as before. If there exists a probabilistic polynomial-time algorithm $\mathcal{A}$ and a polynomial $p(\cdot)$ such that

$$\Pr_{x,r \leftarrow \{0,1\}^{\lambda}}[\mathcal{A}(f(x), r) = \mathrm{gl}(x, r)] \geq \frac{3}{4} + \frac{1}{p(\lambda)}$$

for infinitely many values of λ, then there exists a probabilistic polynomial-time algorithm $\mathcal{A}'$ such that

$$\Pr_{x \leftarrow \{0,1\}^{\lambda}}\left[\mathcal{A}'\left(1^{\lambda}, f(x)\right) \in f^{-1}(f(x))\right] \geq \frac{1}{4 \cdot p(\lambda)}$$

for infinitely many values of λ.

Proof for the More Involved Case

The main observation underlying the proof of this proposition is that for every $r \in \{0,1\}^\lambda$, the values $\mathrm{gl}(x, r \oplus e^i)$ and $\mathrm{gl}(x, r)$ together can be used to compute the ith bit of x.

$$\mathrm{gl}(x,r) \oplus \mathrm{gl}\left(x, r \oplus e^i\right) = \left(\bigoplus_{j=1}^{\lambda} x_j \cdot r_j\right) \oplus \left(\bigoplus_{j=1}^{\lambda} x_j \cdot \left(r_j \oplus e^i_j\right)\right) = x_i \cdot r_i \oplus (x_i \cdot \overline{r_i}) = x_i$$

If $\mathcal{A}$ answers correctly on both $(f(x), r)$ and $(f(x), r \oplus e^i)$, then $\mathcal{A}'$ can correctly compute x_i.

$\mathcal{A}'$ knows only that $\mathcal{A}$ answers correctly with "high" probability.

For this reason, $\mathcal{A}'$ will use multiple random values of r, using each one to obtain an estimate of x_i, and then take the estimate occurring a majority of the time as its final guess for x_i.

As a preliminary step, we show that for many x's the probability that $\mathcal{A}$ answers correctly for both $(f(x), r)$ and $(f(x), r \oplus e^i)$, when r is uniform, is sufficiently high.

Proof for the More Involved Case

Claim Let λ be such that

$$\Pr_{x,r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)] \geq \frac{3}{4} + \frac{1}{p(\lambda)}$$

Then there exists a set $S_\lambda \subseteq \{0,1\}^\lambda$ of size at least $\frac{1}{2p(\lambda)} \cdot 2^\lambda$ such that for every $x \in S_\lambda$ it holds that for every $x \in S_\lambda$, it holds that

$$\Pr_{r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)] \geq \frac{3}{4} + \frac{1}{2p(\lambda)}$$

proof

Let $\varepsilon(\lambda) = 1/p(\lambda)$, and define $S_\lambda \subseteq \{0,1\}^\lambda$ to be the set of all x's for which

$$\Pr_{r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)] \geq \frac{3}{4} + \frac{\varepsilon(\lambda)}{2}$$

Proof for the More Involved Case

Claim Let λ be such that $\Pr_{x,r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)] \geq \frac{3}{4} + \frac{1}{p(\lambda)}$. Then there exists a set $S_\lambda \subseteq \{0,1\}^\lambda$ of size at least $\frac{1}{2p(\lambda)} \cdot 2^\lambda$ such that for every $x \in S_n$ it holds that for every $x \in S_\lambda$, it holds that $\Pr_{r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)] \geq \frac{3}{4} + \frac{1}{2p(\lambda)}$

proof $\Pr_{x,r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)] = \frac{1}{2^\lambda}\sum_{x\in\{0,1\}^\lambda} \Pr_{r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)]$

$$= \frac{1}{2^\lambda}\sum_{x\in S_\lambda} \Pr_{r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)] + \frac{1}{2^\lambda}\sum_{x\notin S_\lambda} \Pr_{r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)]$$

$$\leq \frac{|S_\lambda|}{2^\lambda} + \frac{1}{2^\lambda} \cdot \sum_{x\notin S_\lambda}\left(\frac{3}{4} + \frac{\varepsilon(\lambda)}{2}\right) \leq \frac{|S_\lambda|}{2^\lambda} + \left(\frac{3}{4} + \frac{\varepsilon(\lambda)}{2}\right)$$

$$\frac{3}{4} + \varepsilon(\lambda) \leq \frac{|S_\lambda|}{2^\lambda} + \left(\frac{3}{4} + \frac{\varepsilon(\lambda)}{2}\right)$$

Therefore, $|S_\lambda| \geq \frac{\varepsilon(\lambda)}{2} \cdot 2^\lambda$.

Proof for the More Involved Case

Claim Let λ be such that $\Pr_{x,r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)] \geq \frac{3}{4} + \frac{1}{p(\lambda)}$. Then there exists a set $S_\lambda \subseteq \{0,1\}^\lambda$ of size at least $\frac{1}{2p(\lambda)} \cdot 2^\lambda$ such that for every $x \in S_n$ it holds that for every $x \in S_\lambda$ and every i, it holds that $\Pr_{r\leftarrow\{0,1\}^\lambda}\left[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r) \wedge \mathcal{A}\left(f(x), r \oplus e^i\right) = \mathrm{gl}\left(x, r \oplus e^i\right)\right] \geq \frac{1}{2} + \frac{1}{p(\lambda)}$

proof

Let $\varepsilon(\lambda) = 1/p(\lambda)$, and take S_λ to be the set guaranteed by the previous claim. We have

$$\Pr_{r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) \neq \mathrm{gl}(x,r)] \leq \frac{1}{4} - \frac{\varepsilon(\lambda)}{2}$$

$$\Pr_{r\leftarrow\{0,1\}^\lambda}\left[\mathcal{A}\left(f(x), r \oplus e^i\right) \neq \mathrm{gl}\left(x, r \oplus e^i\right)\right] \leq \frac{1}{4} - \frac{\varepsilon(\lambda)}{2}$$

Union Bound

- $\Pr[E_1 \vee E_2] \leq \Pr[E_1] + \Pr[E_2]$
- $\Pr\left[\bigvee_{i=1}^{k} E_i\right] \leq \sum_{i=1}^{k} \Pr[E_i]$

Proof for the More Involved Case

Claim Let λ be such that $\Pr_{x,r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)] \geq \frac{3}{4} + \frac{1}{p(\lambda)}$. Then there exists a set $S_\lambda \subseteq \{0,1\}^\lambda$ of size at least $\frac{1}{2p(\lambda)} \cdot 2^\lambda$ such that for every $x \in S_n$ it holds that for every $x \in S_\lambda$ and every i, it holds that $\Pr_{r\leftarrow\{0,1\}^\lambda}\left[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r) \wedge \mathcal{A}\left(f(x),r \oplus e^i\right) = \mathrm{gl}\left(x,r \oplus e^i\right)\right] \geq \frac{1}{2} + \frac{1}{p(\lambda)}$

proof

Let $\varepsilon(\lambda) = 1/p(\lambda)$, and take S_λ to be the set guaranteed by the previous claim. We have

$$\Pr_{r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) \neq \mathrm{gl}(x,r)] \leq \frac{1}{4} - \frac{\varepsilon(\lambda)}{2}$$

$$\Pr_{r\leftarrow\{0,1\}^\lambda}\left[\mathcal{A}\left(f(x),r \oplus e^i\right) \neq \mathrm{gl}\left(x,r \oplus e^i\right)\right] \leq \frac{1}{4} - \frac{\varepsilon(\lambda)}{2}$$

The probability that $\mathcal{A}$ is incorrect on either $\mathrm{gl}(x,r)$ or $\mathrm{gl}(x,r \oplus e^i)$ is at most $\frac{1}{2} - \varepsilon(\lambda)$. The probability that both are correct is $\frac{1}{2} + \varepsilon(\lambda)$.

Proof for the More Involved Case

For the rest of the proof we set $\varepsilon(\lambda) = 1/p(\lambda)$ and consider only those values of λ for which

$$\Pr_{x,r \leftarrow \{0,1\}^\lambda}[\mathcal{A}(f(x), r) = \mathrm{gl}(x, r)] \geq \frac{3}{4} + \varepsilon(\lambda)$$

The previous claim states that for an $\varepsilon(n)/2$ fraction of inputs x (a set $S_\lambda \subseteq \{0,1\}^\lambda$ of size at least $\frac{1}{2p(\lambda)} \cdot 2^\lambda$), and any i, algorithm $\mathcal{A}$ answers correctly on both $(f(x), r)$ and $(f(x), r \oplus e^i)$ with probability at least $1/2 + \varepsilon(n)$ over uniform choice of r.

From now on we focus only on such values of x. We construct a probabilistic polynomial-time algorithm $\mathcal{A}'$ that inverts $f(x)$ with probability at least $1/2$ when $x \in S_\lambda$.

$$\Pr_{x \leftarrow \{0,1\}^\lambda}\left[\mathcal{A}'\left(1^\lambda, f(x)\right) \in f^{-1}(f(x))\right]$$

$$\geq \Pr_{x \leftarrow \{0,1\}^\lambda}\left[\mathcal{A}'\left(1^\lambda, f(x)\right) \in f^{-1}(f(x)) \mid x \in S_\lambda\right] \cdot \Pr_{x \leftarrow \{0,1\}^\lambda}[x \in S_\lambda] \geq \frac{1}{4 \cdot p(\lambda)}$$

Proof for the More Involved Case

Proposition Let f and gl be as before. If there exists a probabilistic polynomial-time algorithm $\mathcal{A}$ and a polynomial $p(\cdot)$ such that

$$\Pr_{x,r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)] \geq \frac{3}{4} + \frac{1}{p(\lambda)}$$

for infinitely many values of λ, then there exists a probabilistic polynomial-time algorithm $\mathcal{A}'$ such that

$$\Pr_{x\leftarrow\{0,1\}^\lambda}\left[\mathcal{A}'\left(1^\lambda, f(x)\right) \in f^{-1}(f(x))\right] \geq \frac{1}{4\cdot p(\lambda)}$$

for infinitely many values of λ.

$$\Pr_{x\leftarrow\{0,1\}^\lambda}\left[\mathcal{A}'\left(1^\lambda, f(x)\right) \in f^{-1}(f(x))\right]$$

$$\geq \Pr_{x\leftarrow\{0,1\}^\lambda}\left[\mathcal{A}'\left(1^\lambda, f(x)\right) \in f^{-1}(f(x)) \mid x \in S_\lambda\right] \cdot \Pr_{x\leftarrow\{0,1\}^\lambda}[x \in S_\lambda] \geq \frac{1}{4\cdot p(\lambda)}$$

Proof for the More Involved Case

Algorithm $\mathcal{A}'$, given as input 1^λ and y, works as follows:

1. For $i = 1, \ldots, \lambda$ do:

   Repeatedly choose a uniform $r \in \{0,1\}^\lambda$ and compute $\mathcal{A}(y, r) \oplus \mathcal{A}(y, r \oplus e^i)$ as an "estimate" for the ith bit of the preimage of y. After doing this sufficiently many times, let x_i be the "estimate" that occurs a majority of the time.

2. Output $x = x_1 \cdots x_\lambda$.

By obtaining sufficiently many estimates and letting x_i be the majority value, $\mathcal{A}'$ can ensure that x_i is equal to $\mathrm{gl}(\hat{x}, e^i)$ with probability at least $1 - \frac{1}{2\lambda}$.

Chernoff bound

Proposition Fix $\varepsilon > 0$ and $b \in \{0, 1\}$, and let $\{X_i\}_{i=1,\dots m}$ be independent $0/1$-random variables with $\Pr[X_i = b] = \frac{1}{2} + \varepsilon$ for all i. The probability that their majority value is not b is at most $e^{-\varepsilon^2 m/2}$.

Proof for the More Involved Case

Algorithm $\mathcal{A}'$, given as input 1^λ and y, works as follows:

1. For $i = 1, \ldots, \lambda$ do:

   Repeatedly choose a uniform $r \in \{0, 1\}^\lambda$ and compute $\mathcal{A}(y, r) \oplus \mathcal{A}(y, r \oplus e^i)$ as an "estimate" for the ith bit of the preimage of y. After doing this sufficiently many times, let x_i be the "estimate" that occurs a majority of the time.

2. Output $x = x_1 \cdots x_\lambda$.

By obtaining sufficiently many estimates and letting x_i be the majority value, $\mathcal{A}'$ can ensure that x_i is equal to $\mathrm{gl}(\hat{x}, e^i)$ with probability at least $1 - \frac{1}{2\lambda}$.

- polynomially many estimates suffice

We have that for each i the value x_i computed by $\mathcal{A}'$ is incorrect with probability at most $\frac{1}{2\lambda}$.

A union bound thus shows that $\mathcal{A}'$ is incorrect for some i with probability at most $\lambda \cdot 1/2\lambda = 1/2$, and thus correctly inverts y—with probability at least $1/2$

The Full Proof

Proposition Let f and gl be as before. If there exists a probabilistic polynomial-time algorithm $\mathcal{A}$ and a polynomial $p(\cdot)$ such that

$$\Pr_{x,r\leftarrow\{0,1\}^\lambda}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)] \geq \frac{1}{2} + \frac{1}{p(\lambda)}$$

for infinitely many values of λ, then there exists a probabilistic polynomial-time algorithm $\mathcal{A}'$ and a polynomial $p'(\cdot)$ such that

$$\Pr_{x\leftarrow\{0,1\}^\lambda}\left[\mathcal{A}'\left(1^\lambda, f(x)\right) \in f^{-1}(f(x))\right] \geq \frac{1}{p'(\lambda)}$$

for infinitely many values of λ.

The Full Proof

Proposition Let f and gl be as before. If there exists a probabilistic polynomial-time algorithm $\mathcal{A}$ and a polynomial $p(\cdot)$ such that

$$\Pr_{x,r \leftarrow \{0,1\}^\lambda}[\mathcal{A}(f(x), r) = \mathrm{gl}(x, r)] \geq \frac{1}{2} + \frac{1}{p(\lambda)}$$

for infinitely many values of λ, then there exists a probabilistic polynomial-time algorithm $\mathcal{A}'$ and a polynomial $p'(\cdot)$ such that

$$\Pr_{x \leftarrow \{0,1\}^\lambda}\left[\mathcal{A}'\left(1^\lambda, f(x)\right) \in f^{-1}\big(f(x)\big)\right] \geq \frac{1}{p'(\lambda)}$$

for infinitely many values of λ.

proof Once again we set $\varepsilon(n) = 1/p(\lambda)$ and consider only those values of λ for which $\Pr_{x,r \leftarrow \{0,1\}^\lambda}[\mathcal{A}(f(x), r) = \mathrm{gl}(x, r)] \geq \frac{1}{2} + \varepsilon(\lambda)$

The Full Proof

The following is analogous to the previous claim and is proved in the same way.

Claim Let λ be such that $\Pr_{x,r \leftarrow \{0,1\}^\lambda}[\mathcal{A}(f(x), r) = \mathrm{gl}(x, r)] \geq \frac{1}{2} + \varepsilon(\lambda)$.

Then there exists a set $S_\lambda \subseteq \{0,1\}^\lambda$ of size at least $\frac{\varepsilon(\lambda)}{2} \cdot 2^\lambda$ such that for every $x \in S_\lambda$ it holds that for every $x \in S_\lambda$, it holds that

$$\Pr_{r \leftarrow \{0,1\}^\lambda}[\mathcal{A}(f(x), r) = \mathrm{gl}(x, r)] \geq \frac{1}{2} + \frac{\varepsilon(\lambda)}{2}$$

The Full Proof

If we start by trying to prove an analogue of claim for $\mathcal{A}(f(x), r) = \mathrm{gl}(x, r) \wedge \mathcal{A}\left(f(x), r \oplus e^i\right) = \mathrm{gl}\left(x, r \oplus e^i\right)$, we have for any i

$$\Pr_{r \leftarrow \{0,1\}^\lambda}\left[\mathcal{A}(f(x), r) = \mathrm{gl}(x, r) \wedge \mathcal{A}\left(f(x), r \oplus e^i\right) = \mathrm{gl}\left(x, r \oplus e^i\right)\right] \geq \varepsilon(\lambda)$$

All we can claim is that this estimate will be correct with probability at least $\varepsilon(\lambda)$, which may not be any better than taking a random guess! We cannot claim that flipping the result gives a good estimate, either.

Instead, we design $\mathcal{A}'$ so that it computes $\mathrm{gl}(x, r)$ and $\mathrm{gl}(x, r \oplus e^i)$ by invoking $\mathcal{A}$ only once.

We do this by having $\mathcal{A}'$ run $A(f(x), r \oplus e^i)$, and then simply "guessing" the value $\mathrm{gl}(x, r)$ itself.

The Full Proof

Instead, we design $\mathcal{A}'$ so that it computes $\mathrm{gl}(x, r)$ and $\mathrm{gl}(x, r \oplus e^i)$ by invoking $\mathcal{A}$ only once.

We do this by having $\mathcal{A}'$ run $A(f(x), r \oplus e^i)$, and then simply "guessing" the value $\mathrm{gl}(x, r)$ itself.

The naive way to do this would be to choose the r's independently, as before, and to have $\mathcal{A}'$ make an independent guess of $\mathrm{gl}(x, r)$ for each value of r. But then the probability that all such guesses are correct would be negligible because polynomially many r's are used.

Solution: $\mathcal{A}'$ can generate the r's in a pairwise-independent manner and make its guesses in a particular way so that with non-negligible probability all its guesses are correct.

The Full Proof

Solution: $\mathcal{A}'$ can generate the r's in a pairwise-independent manner and make its guesses in a particular way so that with non-negligible probability all its guesses are correct.

In order to generate m values of r, we have $\mathcal{A}'$ select $\ell = \lceil \log(m+1) \rceil$ independent and uniformly distributed strings $s^1, \dots, s^\ell \in \{0,1\}^\lambda$. Then, for every nonempty subset $I \subseteq \{1, \dots, \ell\}$, we set $r^I = \bigoplus_{i \in I} s^i$.

Since there are $2^\ell - 1$ nonempty subsets, this defines a collection of $2^{\lceil \log(m+1) \rceil} - 1 \geq m$ strings.

The strings are not independent, but they are pairwise independent.

For every two subsets $I \neq J$ there is an index $j \in I \cup J$ such that $j \notin I \cap J$. Without loss of generality, assume $j \notin I$.

Then the value of s^j is uniform and independent of the value of r^I. Since s^j is included in the XOR that defines r^J, this implies that r^J is uniform and independent of r^I as well.

The Full Proof

Solution: $\mathcal{A}'$ can generate the r's in a pairwise-independent manner and make its guesses in a particular way so that with non-negligible probability all its guesses are correct.

In order to generate m values of r, we have $\mathcal{A}'$ select $\ell = \lceil \log(m+1) \rceil$ independent and uniformly distributed strings $s^1, \ldots, s^\ell \in \{0,1\}^\lambda$. Then, for every nonempty subset $I \subseteq \{1, \ldots, \ell\}$, we set $r^I = \bigoplus_{i \in I} s^i$.

We now have the following two important observations:

- Given $\mathrm{gl}(x, s^1), \ldots, \mathrm{gl}(x, s^\ell)$, it is possible to compute $\mathrm{gl}(x, r^I)$ for every subset $I \subseteq \{1, \ldots, \ell\}$. This is because
$$\mathrm{gl}(x, r^I) = \mathrm{gl}\left(x, \bigoplus_{i \in I} s^i\right) = \bigoplus_{i \in I} \mathrm{gl}(x, s^i)$$
- If $\mathcal{A}'$ simply guesses the values of $\mathrm{gl}(x, s^1), \ldots, \mathrm{gl}(x, s^\ell)$ by choosing a uniform bit for each, then all these guesses will be correct with probability $1/2^\ell$. If m is polynomial in the security parameter λ, then $1/2^\ell$ is not negligible, and so with non-negligible probability $\mathcal{A}'$ correctly guesses all the values $\mathrm{gl}(x, s^1), \ldots, \mathrm{gl}(x, s^\ell)$.

The Full Proof

Solution: $\mathcal{A}'$ can generate the r's in a pairwise-independent manner and make its guesses in a particular way so that with non-negligible probability all its guesses are correct.

In order to generate m values of r, we have $\mathcal{A}'$ select $\ell = \lceil \log(m+1) \rceil$ independent and uniformly distributed strings $s^1, \ldots, s^\ell \in \{0,1\}^\lambda$. Then, for every nonempty subset $I \subseteq \{1, \ldots, \ell\}$, we set $r^I = \bigoplus_{i \in I} s^i$.

We can obtain $m = poly(\lambda)$ uniform and pairwise-independent strings $\{r^I\}$ along with correct values for $\{\mathrm{gl}(x, r^I)\}$ with non-negligible probability.

The Full Proof

The full description of an algorithm $\mathcal{A}'$ that receives inputs 1^λ, y and tries to compute an inverse of y as follows:

1. Set $\ell = \lceil \log(m+1) \rceil$, where $m = 2\lambda/\varepsilon(\lambda)^2$.
2. Choose uniform, independent $s^1, \ldots, s^\ell \in \{0,1\}^\ell$ and $\sigma^1, \ldots, \sigma^\ell \in \{0,1\}$.
3. For every nonempty subset $I \subseteq \{1, \ldots, \ell\}$, compute $r^I = \bigoplus_{i \in I} s^i$ and $\sigma^I = \bigoplus_{i \in I} \sigma^i$.
4. For $i = 1, \ldots, \lambda$ do:
    1. For every nonempty subset $I \subseteq \{1, \ldots, \ell\}$, set $x_i^I = \sigma^I \oplus \mathcal{A}(y, r^I \oplus e^i)$.
    2. Set $x_i = \mathrm{majority}_I\{x_i^I\}$
5. Output $x = x_1 \ \cdots \ x_\ell$.

The Full Proof

It remains to compute the probability that $\mathcal{A}'$ outputs $x \in f^{-1}(y)$.

We focus on λ as before, and assume $y = f(\hat{x})$ for some $\hat{x} \in S_\ell$.

As noted earlier, with non-negligible probability all these guesses are correct; we show that conditioned on this event, $\mathcal{A}'$ outputs $x = \hat{x}$ with probability at least $1/2$.

Assume $\sigma_i = \mathrm{gl}(\hat{x}, s^i)$ for all i. Then $\sigma^I = \mathrm{gl}(\hat{x}, r^I)$ for all I. Fix an index $i \in \{1, \ldots, \lambda\}$ and consider the probability that $\mathcal{A}'$ obtains the correct value $x_i = \hat{x}_i$.

For any nonempty I we have $\mathcal{A}(y, r^I \oplus e^i) = \mathrm{gl}(\hat{x}, r^I \oplus e^i)$ with probability at least $1/2 + \varepsilon(\lambda)/2$ over choice of r because $\hat{x} \in S_\lambda$ and $r^I \oplus e^i$ is uniformly distributed.

Thus, for any nonempty subset I we have $\Pr[x_i^I = \hat{x}_i] \geq \frac{1}{2} + \varepsilon(\lambda)/2$.

The Full Proof

It remains to compute the probability that $\mathcal{A}'$ outputs $x \in f^{-1}(y)$.

Thus, for any nonempty subset I we have $\Pr[x_i^I = \hat{x}_i] \geq \frac{1}{2} + \varepsilon(n)/2$.

Moreover, the $\{x_i^I\}_{I \subseteq \{1,\ldots,\ell\}}$ are pairwise independent because the $\{r^I\}_{I \subseteq \{1,\ldots,\ell\}}$ (and hence the $\{r^I \oplus e^i\}_{I \subseteq \{1,\ldots,\ell\}}$) are pairwise independent.

A Tool

Proposition Fix $\varepsilon > 0$ and $b \in \{0, 1\}$, and let $\{X_i\}$ be pairwise-independent, $0/1$-random variables for which $\Pr[X_i = b] \geq 1/2 + \varepsilon$ for all i. Consider the process in which m values $X_1, . . . , X_m$ are recorded and X is set to the value that occurs a strict majority of the time. Then

$$\Pr[X \neq b] \leq \frac{1}{4 \cdot \varepsilon^2 \cdot m}$$

The Full Proof

It remains to compute the probability that $\mathcal{A}'$ outputs $x \in f^{-1}(y)$.

Thus, for any nonempty subset I we have $\Pr[x_i^I = \hat{x}_i] \geq \frac{1}{2} + \varepsilon(n)/2$.

Moreover, the $\left\{x_i^I\right\}_{I \subseteq \{1,\ldots,\ell\}}$ are pairwise independent because the $\{r^I\}_{I \subseteq \{1,\ldots,\ell\}}$ (and hence the $\left\{r^I \oplus e^i\right\}_{I \subseteq \{1,\ldots,\ell\}}$) are pairwise independent.

$$\Pr[x_i \neq \hat{x}_i] \leq \frac{1}{4 \cdot \left(\frac{\varepsilon(\lambda)}{2}\right)^2 \cdot (2^\ell - 1)} \leq \frac{1}{4 \cdot \left(\frac{\varepsilon(\lambda)}{2}\right)^2 \cdot \left(\frac{2\lambda}{\varepsilon(\lambda)^2}\right)} \leq 1/2\lambda$$

The above holds for all i, so by applying a union bound we see that the probability that $x_i \neq \hat{x}_i$ for some i is at most $1/2$. That is, $x_i = \hat{x}_i$ for all i (and hence $x = \hat{x}$) with probability at least $1/2$.

The Full Proof

Putting everything together:

With probability at least $\varepsilon(n)/2$ we have $\hat{x} \in S_\ell$. All the guesses σ_i are correct with probability at least $\frac{1}{2^\ell} \geq \frac{1}{2\cdot\left(\frac{2\lambda}{\varepsilon(\lambda)^2}+1\right)} > \frac{\varepsilon(\lambda)^2}{5\lambda}$ for λ sufficiently large.

Conditioned on both the above, $\mathcal{A}'$ outputs $x = \hat{x}$ with probability at least $1/2$.

The overall probability with which $\mathcal{A}'$ inverts its input is thus at least $\varepsilon(\lambda)^3/20\lambda = 1/(20\,\lambda p(\lambda)^3)$ for infinitely many λ.

$20\,\lambda p(\lambda)^3$ is a polynomial in λ, this proves the proposition.