Manger's Attack revisited

Falko Strenzke[1]

1 - FlexSecure GmbH, Germany,
strenzke@flexsecure.de

February 8, 2013

- RSA-OAEP Encoding introduced to thwart Bleichenbacher's Attack against RSA with PKCS#1 v1.5 Encoding
- The OAEP is a so called CCA2 conversion that secures a cryptosystem against adaptive chosen ciphertext attacks
- (any manipulation of an original ciphertext is detected during the decryption)
- CRYPTO 2001: James Manger introduces a Fault/Timing Attack against straightforward implementations of RSA-OAEP

Manger's Attack

- RSA-OAEP Encoding introduced to thwart Bleichenbacher's Attack against RSA with PKCS#1 v1.5 Encoding
- The OAEP is a so called CCA2 conversion that secures a cryptosystem against adaptive chosen ciphertext attacks
- (any manipulation of an original ciphertext is detected during the decryption)
- CRYPTO 2001: James Manger introduces a Fault/Timing Attack against straightforward implementations of RSA-OAEP

Manger's Attack

- RSA-OAEP Encoding introduced to thwart Bleichenbacher's Attack against RSA with PKCS#1 v1.5 Encoding
- The OAEP is a so called CCA2 conversion that secures a cryptosystem against adaptive chosen ciphertext attacks
- (any manipulation of an original ciphertext is detected during the decryption)
- CRYPTO 2001: James Manger introduces a Fault/Timing Attack against straightforward implementations of RSA-OAEP

Manger's Attack

- RSA-OAEP Encoding introduced to thwart Bleichenbacher's Attack against RSA with PKCS#1 v1.5 Encoding
- The OAEP is a so called CCA2 conversion that secures a cryptosystem against adaptive chosen ciphertext attacks
- (any manipulation of an original ciphertext is detected during the decryption)
- CRYPTO 2001: James Manger introduces a Fault/Timing Attack against straightforward implementations of RSA-OAEP

RSA

- public key: public exponent e and public modulus n
- private key: private exponent d with $x^{ed} = x \mod n$
- encryption: $z = m^e \mod n$
- decryption: $m = z^d = m^{ed} \mod n$

RSA

- public key: public exponent e and public modulus n
- private key: private exponent d with $x^{ed} = x \bmod n$
- encryption: $z = m^e \bmod n$
- decryption: $m = z^d = m^{ed} \bmod n$

- public key: public exponent e and public modulus n
- private key: private exponent d with $x^{ed} = x \mod n$
- encryption: $z = m^e \mod n$
- decryption: $m = z^d = m^{ed} \mod n$

- public key: public exponent e and public modulus n
- private key: private exponent d with $x^{ed} = x \bmod n$
- encryption: $z = m^e \bmod n$
- decryption: $m = z^d = m^{ed} \bmod n$

OAEP Encoding

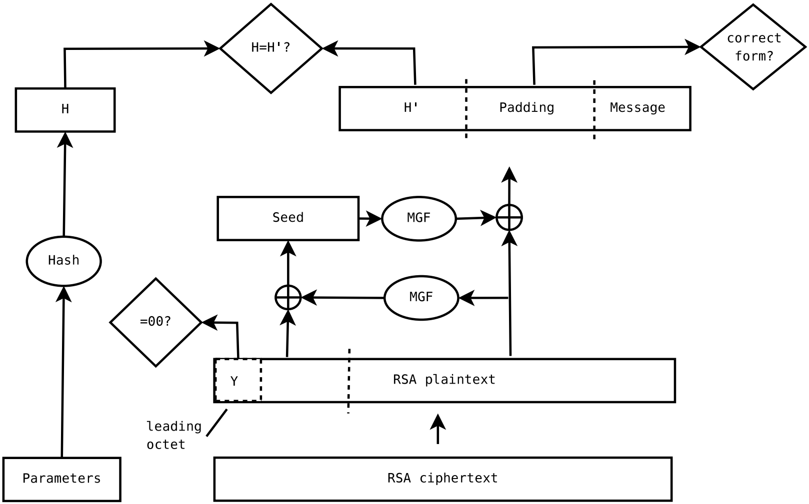
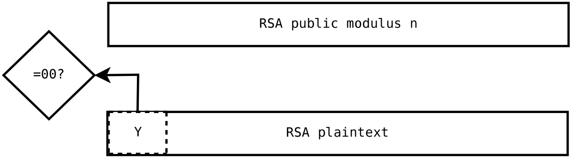
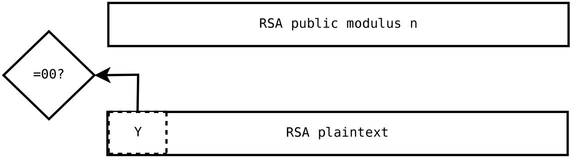


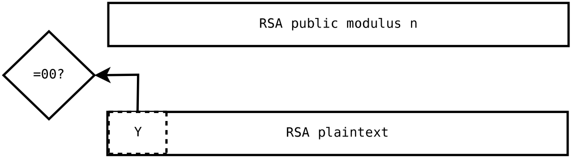
Figure: The RSA-OAEP decoding procedure. Here, ⊕ denotes XOR.



- OAEP Decoding checks that $Y = 0$
- ($Y \neq 0 \rightarrow$ "supernumerary octet")
- $Y \neq 0$ can be learned either through
  - a specific error message
  - shorter time to the error message compared to later OAEP errors
  - (time difference might become **huge** if the attacker can control the public parameters to be hashed within the OAEP decoding routine)

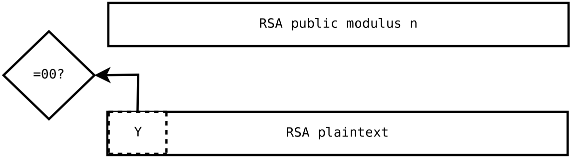


- OAEP Decoding checks that $Y = 0$
- ($Y \neq 0 \rightarrow$ "supernumerary octet")
- $Y \neq 0$ can be learned either through
  - a specific error message
  - shorter time to the error message compared to later OAEP errors
  - (time difference might become **huge** if the attacker can control the public parameters to be hashed within the OAEP decoding routine)



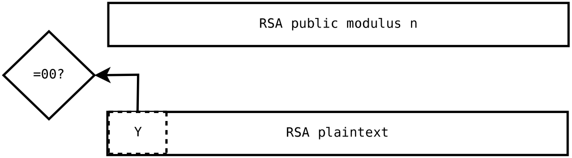
- OAEP Decoding checks that $Y = 0$
- ($Y \neq 0 \rightarrow$ "supernumerary octet")
- $Y \neq 0$ can be learned either through
  - a specific error message
  - shorter time to the error message compared to later OAEP errors
  - (time difference might become **huge** if the attacker can control the public parameters to be hashed within the OAEP decoding routine)

Manger's Attack - the observable Error Condition



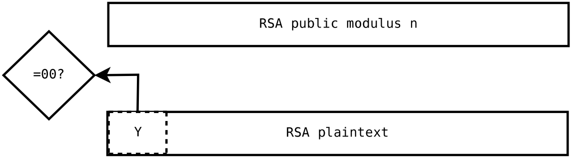
- OAEP Decoding checks that $Y = 0$
- $(Y \neq 0 \rightarrow$ "supernumerary octet")
- $Y \neq 0$ can be learned either through
  - a specific error message
  - shorter time to the error message compared to later OAEP errors
  - (time difference might become **huge** if the attacker can control the public parameters to be hashed within the OAEP decoding routine)

Manger's Attack - the observable Error Condition



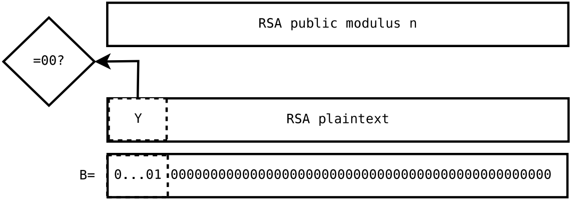
- OAEP Decoding checks that $Y = 0$
- ($Y \neq 0 \rightarrow$ "supernumerary octet")
- $Y \neq 0$ can be learned either through
  - a specific error message
  - shorter time to the error message compared to later OAEP errors
  - (time difference might become **huge** if the attacker can control the public parameters to be hashed within the OAEP decoding routine)

Manger's Attack - the observable Error Condition

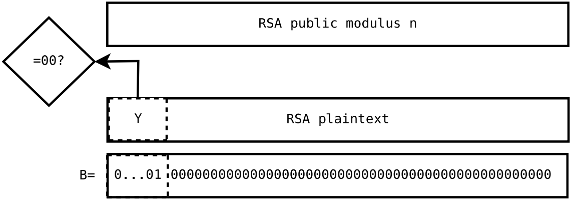


- OAEP Decoding checks that $Y = 0$
- ($Y \neq 0 \rightarrow$ "supernumerary octet")
- $Y \neq 0$ can be learned either through
  - a specific error message
  - shorter time to the error message compared to later OAEP errors
  - (time difference might become **huge** if the attacker can control the public parameters to be hashed within the OAEP decoding routine)

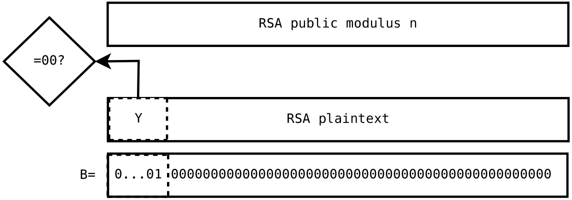
Manger's Attack - the Information Gain



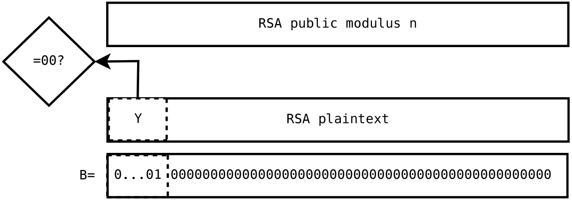
- The attacker wants to decrypt the ciphertext $c_0 = m_0^e \bmod n$
- He chooses $f \in \{0, 1, \ldots, n-1\}$
- He creates ciphertexts $c_f = f^e c_0 = (fm_0)^e \bmod n$
- He observes the decryption of c_f
- If $Y \neq 0$ he learns $\boxed{fm_0 \bmod n \geq B}$
- Manger gives a specific strategy how to choose f initially
- and how to adapt f in in subsequent queries



- The attacker wants to decrypt the ciphertext $c_0 = m_0^e \bmod n$
- He chooses $f \in \{0, 1, \ldots, n-1\}$
- He creates ciphertexts $c_f = f^e c_0 = (fm_0)^e \bmod n$
- He observes the decryption of c_f
- If $Y \neq 0$ he learns $\boxed{fm_0 \bmod n \geq B}$
- Manger gives a specific strategy how to choose f initially
- and how to adapt f in in subsequent queries

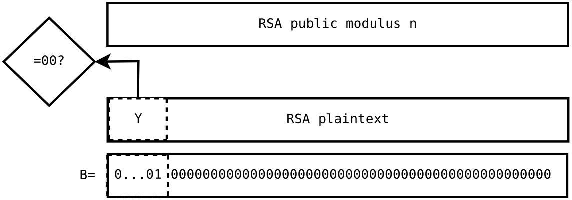


- The attacker wants to decrypt the ciphertext $c_0 = m_0^e \bmod n$
- He chooses $f \in \{0, 1, \ldots, n-1\}$
- He creates ciphertexts $c_f = f^e c_0 = (fm_0)^e \bmod n$
- He observes the decryption of c_f
- If $Y \neq 0$ he learns $\boxed{fm_0 \bmod n \geq B}$
- Manger gives a specific strategy how to choose f initially
- and how to adapt f in in subsequent queries

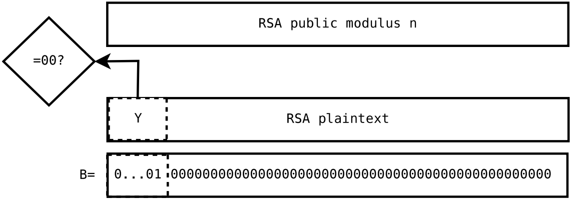


- The attacker wants to decrypt the ciphertext $c_0 = m_0^e \bmod n$
- He chooses $f \in \{0, 1, \ldots, n-1\}$
- He creates ciphertexts $c_f = f^e c_0 = (fm_0)^e \bmod n$
- He observes the decryption of c_f
- If $Y \neq 0$ he learns $\boxed{fm_0 \bmod n \geq B}$
- Manger gives a specific strategy how to choose f initially
- and how to adapt f in in subsequent queries

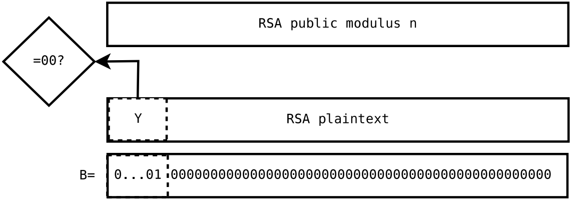
Manger's Attack - the Information Gain



- The attacker wants to decrypt the ciphertext $c_0 = m_0^e \bmod n$
- He chooses $f \in \{0, 1, \ldots, n-1\}$
- He creates ciphertexts $c_f = f^e c_0 = (fm_0)^e \bmod n$
- He observes the decryption of c_f
- If $Y \neq 0$ he learns $\boxed{fm_0 \bmod n \geq B}$
- Manger gives a specific strategy how to choose f initially
- and how to adapt f in in subsequent queries



- The attacker wants to decrypt the ciphertext $c_0 = m_0^e \bmod n$
- He chooses $f \in \{0, 1, \ldots, n-1\}$
- He creates ciphertexts $c_f = f^e c_0 = (fm_0)^e \bmod n$
- He observes the decryption of c_f
- If $Y \neq 0$ he learns $\boxed{fm_0 \bmod n \geq B}$
- Manger gives a specific strategy how to choose f initially
- and how to adapt f in in subsequent queries



- The attacker wants to decrypt the ciphertext $c_0 = m_0^e \bmod n$
- He chooses $f \in \{0, 1, \ldots, n-1\}$
- He creates ciphertexts $c_f = f^e c_0 = (fm_0)^e \bmod n$
- He observes the decryption of c_f
- If $Y \neq 0$ he learns $\boxed{fm_0 \bmod n \geq B}$
- Manger gives a specific strategy how to choose f initially
- and how to adapt f in in subsequent queries

```
lzero = num - flen;
if (lzero < 0)
{
  /* signalling this error immediately after detection might allow for
  * side-channel attacks (e.g. timing if 'plen' is huge – cf. James
  * H. Manger, "A Chosen Ciphertext Attack on RSA Optimal
  * Asymmetric Encryption Padding (OAEP) [...]", CRYPTO 2001),
  * so we use a 'bad' flag */
  bad = 1;
  lzero = 0;
  flen = num; /* don't overflow the memcpy to padded_from */
}
...
if (memcmp(db, phash, SHA_DIGEST_LENGTH) != 0 || bad)
  goto decoding_err;
```

```
...
key_length /= 8;
if(in_length > key_length)
  throw Decoding_Error("Invalid EME1 encoding");
SecureVector<byte> tmp(key_length);
tmp.copy(key_length - in_length, in, in_length);
mgf->mask(tmp + HASH_LENGTH, tmp.size() - HASH_LENGTH, tmp,
HASH_LENGTH);
mgf->mask(tmp, HASH_LENGTH, tmp + HASH_LENGTH, tmp.size() -
HASH_LENGTH);
for(u32bit j = 0; j != Phash.size(); ++j)
  if(tmp[j+HASH_LENGTH] != Phash[j])
    throw Decoding_Error("Invalid EME1 encoding");
...
```

- the strongest form of Manger's Attack (exploiting the running time of hash computation of huge Parameters) is not possible for either library
- OpenSSL did not respond to the report of the potential vulnerability
- The Botan main developer released a patch after the vulnerability was reported to him

Analysis of OpenSSL and Botan

- the strongest form of Manger's Attack (exploiting the running time of hash computation of huge Parameters) is not possible for either library
- OpenSSL did not respond to the report of the potential vulnerability
- The Botan main developer released a patch after the vulnerability was reported to him

Analysis of OpenSSL and Botan

- the strongest form of Manger's Attack (exploiting the running time of hash computation of huge Parameters) is not possible for either library
- OpenSSL did not respond to the report of the potential vulnerability
- The Botan main developer released a patch after the vulnerability was reported to him

A new potential Vulnerability in the Integer to Octet String Conversion

```
void BigInt::binary_encode(byte output[]) const
{
  const u32bit sig_bytes = bytes();
  for(u32bit j = 0; j != sig_bytes; ++j)
    output[sig_bytes-j-1] = byte_at(j);
}
```

- the running time of this routine obviously depends on the number of octets of the encoded integer
- → potential timing or power vulnerability!
- independent of encoding method
- the integer encoding routine in OpenSSL is equivalent

A new potential Vulnerability in the Integer to Octet String Conversion

```
void BigInt::binary_encode(byte output[]) const
{
  const u32bit sig_bytes = bytes();
  for(u32bit j = 0; j != sig_bytes; ++j)
    output[sig_bytes-j-1] = byte_at(j);
}
```

- the running time of this routine obviously depends on the number of octets of the encoded integer
- → potential timing or power vulnerability!
- independent of encoding method
- the integer encoding routine in OpenSSL is equivalent

A new potential Vulnerability in the Integer to Octet String Conversion

```
void BigInt::binary_encode(byte output[]) const
{
  const u32bit sig_bytes = bytes();
  for(u32bit j = 0; j != sig_bytes; ++j)
    output[sig_bytes-j-1] = byte_at(j);
}
```

- the running time of this routine obviously depends on the number of octets of the encoded integer
- → potential timing or power vulnerability!
- independent of encoding method
- the integer encoding routine in OpenSSL is equivalent

A new potential Vulnerability in the Integer to Octet String Conversion

```
void BigInt::binary_encode(byte output[]) const
{
  const u32bit sig_bytes = bytes();
  for(u32bit j = 0; j != sig_bytes; ++j)
    output[sig_bytes-j-1] = byte_at(j);
}
```

- the running time of this routine obviously depends on the number of octets of the encoded integer
- $\rightarrow$ potential timing or power vulnerability!
- independent of encoding method
- the integer encoding routine in OpenSSL is equivalent

A potential Vulnerability in the Multi-Precision Integer (MPI) Arithmetic

- We take a look back one step further from the integer encoding routine
- with respect to conditional branching based on $Y = 0$
- We choose the PolarSSL Library for embedded systems
- We assume the last operation of the RSA computation to be a modular reduction implemented as a division
- in PolarSSL, the result of the division is copied with routine `mpi_copy()`

A potential Vulnerability in the Multi-Precision Integer (MPI) Arithmetic

- We take a look back one step further from the integer encoding routine
- with respect to conditional branching based on $Y = 0$
- We choose the PolarSSL Library for embedded systems
- We assume the last operation of the RSA computation to be a modular reduction implemented as a division
- in PolarSSL, the result of the division is copied with routine `mpi_copy()`

A potential Vulnerability in the Multi-Precision Integer (MPI) Arithmetic

- We take a look back one step further from the integer encoding routine
- with respect to conditional branching based on $Y = 0$
- We choose the PolarSSL Library for embedded systems
- We assume the last operation of the RSA computation to be a modular reduction implemented as a division
- in PolarSSL, the result of the division is copied with routine `mpi_copy()`

A potential Vulnerability in the Multi-Precision Integer (MPI) Arithmetic

- We take a look back one step further from the integer encoding routine
- with respect to conditional branching based on $Y = 0$
- We choose the PolarSSL Library for embedded systems
- We assume the last operation of the RSA computation to be a modular reduction implemented as a division
- in PolarSSL, the result of the division is copied with routine `mpi_copy()`

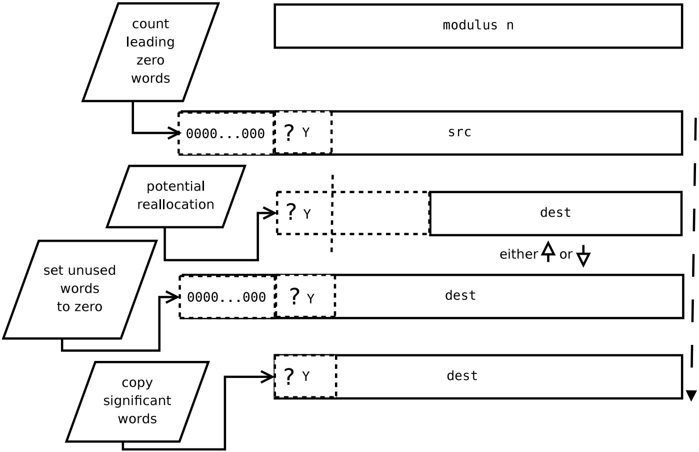
A potential Vulnerability in the Multi-Precision Integer (MPI) Arithmetic

- We take a look back one step further from the integer encoding routine
- with respect to conditional branching based on $Y = 0$
- We choose the PolarSSL Library for embedded systems
- We assume the last operation of the RSA computation to be a modular reduction implemented as a division
- in PolarSSL, the result of the division is copied with routine `mpi_copy()`

The mpi_copy() Routine in the PolarSSL Library

```
typedef struct {
int n;
U8 *p;
} mpi;
int mpi_copy( mpi *X, const mpi *Z ) { // Z is src
  int ret, i;
  if( X == Z )
   return( 0 );
  for( i = Z->n - 1; i > 0; i - - )
    if( Z->p[i] != 0 )
      break;
  i++; // i = # significant words in Z (src)
  X->s = Z->s;
  MPI_CHK( mpi_grow( X, i ) );
  memset( X->p, 0, X->n * ciL );
  memcpy( X->p, Z->p,    i* ciL );
  . . .
}
```

The mpi_copy() Routine in the PolarSSL Library



- the call to memcpy (potentially) offers a plain dependency of the running time on "$Y = 0$?"
- other routines in this function also show such dependencies
- (also with opposed timing effects regarding $Y = 0$)
- but depend on the history of source and destination MPI operands
- $\rightarrow$ must be accounted for in a concrete implementation

The mpi_copy() Routine in the PolarSSL Library

- the call to memcpy (potentially) offers a plain dependency of the running time on "$Y = 0$?"
- other routines in this function also show such dependencies
- (also with opposed timing effects regarding $Y = 0$)
- but depend on the history of source and destination MPI operands
- $\rightarrow$ must be accounted for in a concrete implementation

The mpi_copy() Routine in the PolarSSL Library

- the call to memcpy (potentially) offers a plain dependency of the running time on "$Y = 0$?"
- other routines in this function also show such dependencies
- (also with opposed timing effects regarding $Y = 0$)
- but depend on the history of source and destination MPI operands
- $\rightarrow$ must be accounted for in a concrete implementation

- the call to memcpy (potentially) offers a plain dependency of the running time on "$Y = 0$?"
- other routines in this function also show such dependencies
- (also with opposed timing effects regarding $Y = 0$)
- but depend on the history of source and destination MPI operands
- $\rightarrow$ must be accounted for in a concrete implementation

The mpi_copy() Routine in the PolarSSL Library

- the call to memcpy (potentially) offers a plain dependency of the running time on "$Y = 0$?"
- other routines in this function also show such dependencies
- (also with opposed timing effects regarding $Y = 0$)
- but depend on the history of source and destination MPI operands
- $\rightarrow$ must be accounted for in a concrete implementation

Impact of the keysize on the MPI related Vulnerability

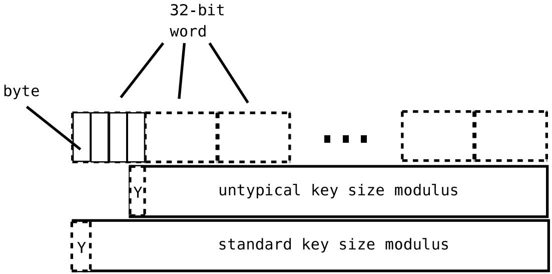
- RSA key size: bit length of the public modulus n
- typical key sizes are multiples of 32 (powers of two)
- with untypical keysizes the MPI related vulnerabilities are also possible with 32-bit words

Impact of the keysize on the MPI related Vulnerability

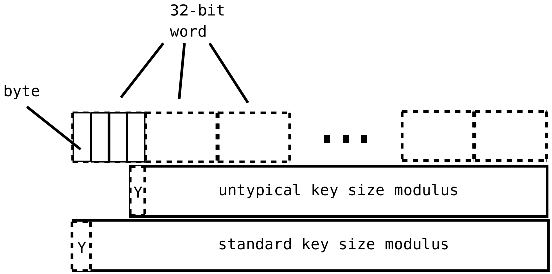
- RSA key size: bit length of the public modulus n
- typical key sizes are multiples of 32 (powers of two)
- with untypical keysizes the MPI related vulnerabilities are also possible with 32-bit words

- RSA key size: bit length of the public modulus n
- typical key sizes are multiples of 32 (powers of two)
- with untypical keysizes the MPI related vulnerabilities are also possible with 32-bit words

Impact of the keysize on the MPI related Vulnerability



- for such untypical key sizes $Y = 0$ means that the number of words in m is smaller by one compared to $Y \neq 0$



- for such untypical key sizes $Y = 0$ means that the number of words in m is smaller by one compared to $Y \neq 0$

On the relevance of the new potential Vulnerabilities

- we have identified "unbalanced conditional branching" based on a message property
- this gives an onset for timing attacks (TA)
- and simple power analysis attacks (SPA) (refined TA revealing the running time of individual subroutines)
- from the point of view of security engineering, any implementation must analyzed with respect to these vulnerabilities

- we have identified "unbalanced conditional branching" based on a message property
- this gives an onset for timing attacks (TA)
- and simple power analysis attacks (SPA) (refined TA revealing the running time of individual subroutines)
- from the point of view of security engineering, any implementation must analyzed with respect to these vulnerabilities

On the relevance of the new potential Vulnerabilities

- we have identified "unbalanced conditional branching" based on a message property
- this gives an onset for timing attacks (TA)
- and simple power analysis attacks (SPA) (refined TA revealing the running time of individual subroutines)
- from the point of view of security engineering, any implementation must analyzed with respect to these vulnerabilities

On the relevance of the new potential Vulnerabilities

- we have identified "unbalanced conditional branching" based on a message property
- this gives an onset for timing attacks (TA)
- and simple power analysis attacks (SPA) (refined TA revealing the running time of individual subroutines)
- from the point of view of security engineering, any implementation must analyzed with respect to these vulnerabilities

platform properties influencing the exploitability:

- source code
- hardware
- compiler
- "accessibility" for an attacker (timing / power)

On the relevance of these potential Vulnerabilities

platform properties influencing the exploitability:

- source code
- hardware
- compiler
- "accessibility" for an attacker (timing / power)

On the relevance of these potential Vulnerabilities

platform properties influencing the exploitability:

- source code
- hardware
- compiler
- “accessibility” for an attacker (timing / power)

On the relevance of these potential Vulnerabilities

platform properties influencing the exploitability:

- source code
- hardware
- compiler
- "accessibility" for an attacker (timing / power)

On the relevance of these potential Vulnerabilities

platform properties influencing the exploitability:

- source code $\leftarrow$ solve problem here for TA
- hardware
- compiler
- "accessibility" for an attacker (timing / power)

- Previously proposed countermeasures incurr security threats:
- (1) if $Y \neq 0$, one shall used randomly generated dummy values in the further OAEP decoding
- $\rightarrow$ threat: random values turn an otherwise deterministic processing indeterministic, which might be detected through side channels by repeatedly decrypting the same ciphertext
- (2) if $Y \neq 0$, one shall set the $m = 0 \ldots 0$ in the further OAEP decoding
- $\rightarrow$ threat: an "all zero" octet string is an extreme case of low Hamming weight and might very likely be detected through power analysis

Previously proposed Countermeasures

- Previously proposed countermeasures incurr security threats:
- (1) if $Y \neq 0$, one shall used randomly generated dummy values in the further OAEP decoding
- $\rightarrow$ threat: random values turn an otherwise deterministic processing indeterministic, which might be detected through side channels by repeatedly decrypting the same ciphertext
- (2) if $Y \neq 0$, one shall set the $m = 0 \ldots 0$ in the further OAEP decoding
- $\rightarrow$ threat: an "all zero" octet string is an extreme case of low Hamming weight and might very likely be detected through power analysis

Previously proposed Countermeasures

- Previously proposed countermeasures incurr security threats:
- (1) if $Y \neq 0$, one shall used randomly generated dummy values in the further OAEP decoding
- $\rightarrow$ threat: random values turn an otherwise deterministic processing indeterministic, which might be detected through side channels by repeatedly decrypting the same ciphertext
- (2) if $Y \neq 0$, one shall set the $m = 0 \ldots 0$ in the further OAEP decoding
- $\rightarrow$ threat: an "all zero" octet string is an extreme case of low Hamming weight and might very likely be detected through power analysis

Previously proposed Countermeasures

- Previously proposed countermeasures incurr security threats:
- (1) if $Y \neq 0$, one shall used randomly generated dummy values in the further OAEP decoding
- $\rightarrow$ threat: random values turn an otherwise deterministic processing indeterministic, which might be detected through side channels by repeatedly decrypting the same ciphertext
- (2) if $Y \neq 0$, one shall set the $m = 0 \ldots 0$ in the further OAEP decoding
- $\rightarrow$ threat: an "all zero" octet string is an extreme case of low Hamming weight and might very likely be detected through power analysis

Previously proposed Countermeasures

- Previously proposed countermeasures incurr security threats:
- (1) if $Y \neq 0$, one shall used randomly generated dummy values in the further OAEP decoding
- $\rightarrow$ threat: random values turn an otherwise deterministic processing indeterministic, which might be detected through side channels by repeatedly decrypting the same ciphertext
- (2) if $Y \neq 0$, one shall set the $m = 0 \dots 0$ in the further OAEP decoding
- $\rightarrow$ threat: an "all zero" octet string is an extreme case of low Hamming weight and might very likely be detected through power analysis

- We give a countermeasure against the MPI encoding routine:
- C++ source code
- number of iterations in the encoding routine depends only on the key size
- enforces $Y = 0$ already in the encoding routine
- uses the `volatile` specifier to take away the compilers ability to remove unnecessary operations
- use no conditional branching, not even comparison operators
- but only logical operations
- logical masking replaces conditional branching

Effective Countermeasures against Timing Attacks

- We give a countermeasure against the MPI encoding routine:
- C++ source code
- number of iterations in the encoding routine depends only on the key size
- enforces $Y = 0$ already in the encoding routine
- uses the `volatile` specifier to take away the compilers ability to remove unnecessary operations
- use no conditional branching, not even comparison operators
- but only logical operations
- logical masking replaces conditional branching

Effective Countermeasures against Timing Attacks

- We give a countermeasure against the MPI encoding routine:
- C++ source code
- number of iterations in the encoding routine depends only on the key size
- enforces $Y = 0$ already in the encoding routine
- uses the `volatile` specifier to take away the compilers ability to remove unnecessary operations
- use no conditional branching, not even comparison operators
- but only logical operations
- logical masking replaces conditional branching

Effective Countermeasures against Timing Attacks

- We give a countermeasure against the MPI encoding routine:
- C++ source code
- number of iterations in the encoding routine depends only on the key size
- enforces $Y = 0$ already in the encoding routine
- uses the `volatile` specifier to take away the compilers ability to remove unnecessary operations
- use no conditional branching, not even comparison operators
- but only logical operations
- logical masking replaces conditional branching

Effective Countermeasures against Timing Attacks

- We give a countermeasure against the MPI encoding routine:
- C++ source code
- number of iterations in the encoding routine depends only on the key size
- enforces $Y = 0$ already in the encoding routine
- uses the `volatile` specifier to take away the compilers ability to remove unnecessary operations
- use no conditional branching, not even comparison operators
- but only logical operations
- logical masking replaces conditional branching

Effective Countermeasures against Timing Attacks

- We give a countermeasure against the MPI encoding routine:
- C++ source code
- number of iterations in the encoding routine depends only on the key size
- enforces $Y = 0$ already in the encoding routine
- uses the `volatile` specifier to take away the compilers ability to remove unnecessary operations
- use no conditional branching, not even comparison operators
- but only logical operations
- logical masking replaces conditional branching

Effective Countermeasures against Timing Attacks

- We give a countermeasure against the MPI encoding routine:
- C++ source code
- number of iterations in the encoding routine depends only on the key size
- enforces $Y = 0$ already in the encoding routine
- uses the `volatile` specifier to take away the compilers ability to remove unnecessary operations
- use no conditional branching, not even comparison operators
- but only logical operations
- logical masking replaces conditional branching

Effective Countermeasures against Timing Attacks

- We give a countermeasure against the MPI encoding routine:
- C++ source code
- number of iterations in the encoding routine depends only on the key size
- enforces $Y = 0$ already in the encoding routine
- uses the `volatile` specifier to take away the compilers ability to remove unnecessary operations
- use no conditional branching, not even comparison operators
- but only logical operations
- logical masking replaces conditional branching

- The last MPI routines in the decryption must “hide” the number of words of m
- this can be done in the same manner as protecting the the MPI encoding routine

Outline of Countermeasures for the MPI Arithmetic

- The last MPI routines in the decryption must "hide" the number of words of m
- this can be done in the same manner as protecting the the MPI encoding routine

Conclusion

- concerning the OpenSSL countermeasure, it is obvious that there is no common notion concerning the relevance of the leakage of “small” timing differences
- (compare with cache-timing attacks against AES, where minimal timing differences are regarded as critical)
- even though Manger’s Attack is known for almost 10 years, we could find new leakages about crucial properties of the message
  - in the MPI encoding routines
  - in the MPI arithmetic (under certain circumstances)
- we propose countermeasures that ensure running times only dependent on the key size for the potentially vulnerable routines

Conclusion

- concerning the OpenSSL countermeasure, it is obvious that there is no common notion concerning the relevance of the leakage of "small" timing differences
- (compare with cache-timing attacks against AES, where minimal timing differences are regarded as critical)
- even though Manger's Attack is known for almost 10 years, we could find new leakages about crucial properties of the message
  - in the MPI encoding routines
  - in the MPI arithmetic (under certain circumstances)
- we propose countermeasures that ensure running times only dependent on the key size for the potentially vulnerable routines

Conclusion

- concerning the OpenSSL countermeasure, it is obvious that there is no common notion concerning the relevance of the leakage of "small" timing differences
- (compare with cache-timing attacks against AES, where minimal timing differences are regarded as critical)
- even though Manger's Attack is known for almost 10 years, we could find new leakages about crucial properties of the message
  - in the MPI encoding routines
  - in the MPI arithmetic (under certain circumstances)
- we propose countermeasures that ensure running times only dependent on the key size for the potentially vulnerable routines

Conclusion

- concerning the OpenSSL countermeasure, it is obvious that there is no common notion concerning the relevance of the leakage of “small” timing differences
- (compare with cache-timing attacks against AES, where minimal timing differences are regarded as critical)
- even though Manger’s Attack is known for almost 10 years, we could find new leakages about crucial properties of the message
  - in the MPI encoding routines
  - in the MPI arithmetic (under certain circumstances)
- we propose countermeasures that ensure running times only dependent on the key size for the potentially vulnerable routines

Conclusion

- concerning the OpenSSL countermeasure, it is obvious that there is no common notion concerning the relevance of the leakage of "small" timing differences
- (compare with cache-timing attacks against AES, where minimal timing differences are regarded as critical)
- even though Manger's Attack is known for almost 10 years, we could find new leakages about crucial properties of the message
  - in the MPI encoding routines
  - in the MPI arithmetic (under certain circumstances)
- we propose countermeasures that ensure running times only dependent on the key size for the potentially vulnerable routines

Conclusion

- concerning the OpenSSL countermeasure, it is obvious that there is no common notion concerning the relevance of the leakage of “small” timing differences
- (compare with cache-timing attacks against AES, where minimal timing differences are regarded as critical)
- even though Manger’s Attack is known for almost 10 years, we could find new leakages about crucial properties of the message
  - in the MPI encoding routines
  - in the MPI arithmetic (under certain circumstances)
- we propose countermeasures that ensure running times only dependent on the key size for the potentially vulnerable routines

- Thank You!