

Message-aimed Side Channel and Fault Attacks against Public Key Cryptosystems with homomorphic Properties

Falko Strenzke

FlexSecure GmbH, Germany,
strenzke@flexsecure.de

February 20, 2011

- We will take a look at a certain type of side channel attack against public key cryptosystems (PKC)
- which require
 - the PKC to have a homomorphic property:
$$\mathcal{E}(a) * \mathcal{E}(b) = \mathcal{E}(a \oplus b)$$
 - the implementation to reveal a certain property of the plaintext
- which aim at recovering the message to certain ciphertext
- are conducted as (adaptively) chosen-ciphertext attacks
- We will consider the RSA and McEliece cryptosystems
 - learn about new resp. recent results
 - *compare the results for both PKCs*

- We will take a look at a certain type of side channel attack against public key cryptosystems (PKC)
- which require
 - the PKC to have a homomorphic property:
$$\mathcal{E}(a) \bullet \mathcal{E}(b) = \mathcal{E}(a \odot b)$$
 - the implementation to reveal a certain property of the plaintext
- which aim at recovering the message to certain ciphertext
- are conducted as (adaptively) chosen-ciphertext attacks
- We will consider the RSA and McEliece cryptosystems
 - learn about new resp. recent results
 - *compare the results for both PKCs*

- We will take a look at a certain type of side channel attack against public key cryptosystems (PKC)
- which require
 - the PKC to have a homomorphic property:
$$\mathcal{E}(a) \bullet \mathcal{E}(b) = \mathcal{E}(a \odot b)$$
 - the implementation to reveal a certain property of the plaintext
- which aim at recovering the message to certain ciphertext
- are conducted as (adaptively) chosen-ciphertext attacks
- We will consider the RSA and McEliece cryptosystems
 - learn about new resp. recent results
 - compare the results for both PKCs

- We will take a look at a certain type of side channel attack against public key cryptosystems (PKC)
- which require
 - the PKC to have a homomorphic property:
$$\mathcal{E}(a) \bullet \mathcal{E}(b) = \mathcal{E}(a \odot b)$$
 - the implementation to reveal a certain property of the plaintext
- which aim at recovering the message to certain ciphertext
- are conducted as (adaptively) chosen-ciphertext attacks
- We will consider the RSA and McEliece cryptosystems
 - learn about new resp. recent results
 - compare the results for both PKCs

- We will take a look at a certain type of side channel attack against public key cryptosystems (PKC)
- which require
 - the PKC to have a homomorphic property:
$$\mathcal{E}(a) \bullet \mathcal{E}(b) = \mathcal{E}(a \odot b)$$
 - the implementation to reveal a certain property of the plaintext
- which aim at recovering the message to certain ciphertext
- are conducted as (adaptively) chosen-ciphertext attacks
- We will consider the RSA and McEliece cryptosystems
 - learn about new resp. recent results
 - compare the results for both PKCs

- We will take a look at a certain type of side channel attack against public key cryptosystems (PKC)
- which require
 - the PKC to have a homomorphic property:
$$\mathcal{E}(a) \bullet \mathcal{E}(b) = \mathcal{E}(a \odot b)$$
 - the implementation to reveal a certain property of the plaintext
- which aim at recovering the message to certain ciphertext
- are conducted as (adaptively) chosen-ciphertext attacks
- We will consider the RSA and McEliece cryptosystems
 - learn about new resp. recent results
 - compare the results for both PKCs

- We will take a look at a certain type of side channel attack against public key cryptosystems (PKC)
- which require
 - the PKC to have a homomorphic property:
$$\mathcal{E}(a) \bullet \mathcal{E}(b) = \mathcal{E}(a \odot b)$$
 - the implementation to reveal a certain property of the plaintext
- which aim at recovering the message to certain ciphertext
- are conducted as (adaptively) chosen-ciphertext attacks
- We will consider the RSA and McEliece cryptosystems
 - learn about new resp. recent results
 - *compare the results for both PKCs*

- We will take a look at a certain type of side channel attack against public key cryptosystems (PKC)
- which require
 - the PKC to have a homomorphic property:
$$\mathcal{E}(a) \bullet \mathcal{E}(b) = \mathcal{E}(a \odot b)$$
 - the implementation to reveal a certain property of the plaintext
- which aim at recovering the message to certain ciphertext
- are conducted as (adaptively) chosen-ciphertext attacks
- We will consider the RSA and McEliece cryptosystems
 - learn about new resp. recent results
 - *compare the results for both PKCs*

- We will take a look at a certain type of side channel attack against public key cryptosystems (PKC)
- which require
 - the PKC to have a homomorphic property:
$$\mathcal{E}(a) \bullet \mathcal{E}(b) = \mathcal{E}(a \odot b)$$
 - the implementation to reveal a certain property of the plaintext
- which aim at recovering the message to certain ciphertext
- are conducted as (adaptively) chosen-ciphertext attacks
- We will consider the RSA and McEliece cryptosystems
 - learn about new resp. recent results
 - *compare the results for both PKCs*

- 1 Introduction
- 2 Decryption Oracle Attacks against the RSA Cryptosystem
- 3 Decryption Oracle Attacks against the McEliece Cryptosystem
- 4 Comparison Between the Attacks against McEliece and RSA
- 5 Countermeasures
- 6 Conclusion

- 1 Introduction
- 2 Decryption Oracle Attacks against the RSA Cryptosystem
- 3 Decryption Oracle Attacks against the McEliece Cryptosystem
- 4 Comparison Between the Attacks against McEliece and RSA
- 5 Countermeasures
- 6 Conclusion

- 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-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-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-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

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

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

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

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

OAEP Encoding

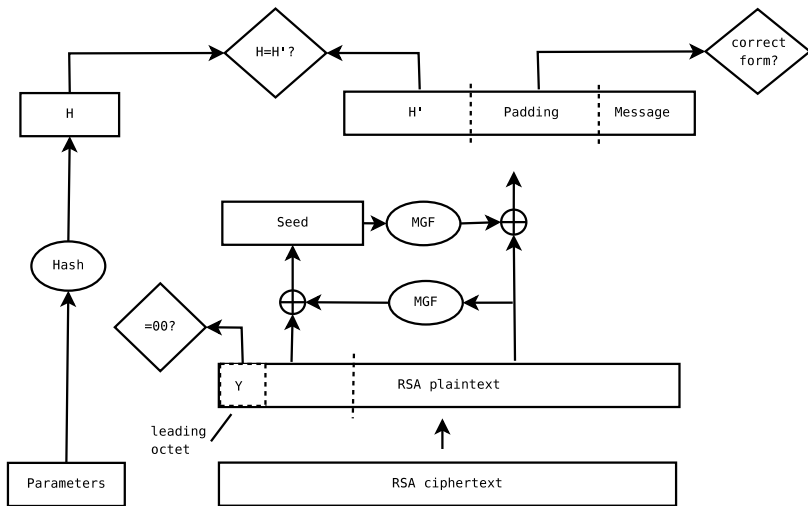
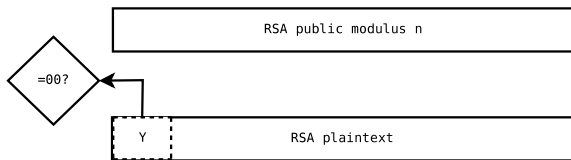


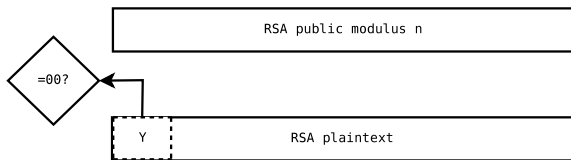
Figure: The RSA-OAEP decoding procedure. Here, \oplus denotes XOR.

Manger's Attack - the observable Error Condition



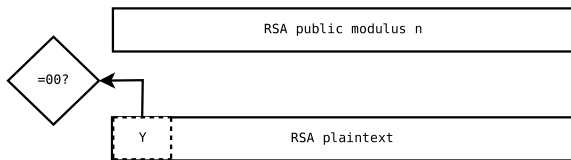
- OAEP Decoding checks that $Y = 0$
- ($Y \neq 0 \rightarrow$ “supernumerary octet”)
- $Y \neq 0$ can be learned either through
 - a specific error message
 - a shorter time to the error message compared to later OAEP errors

Manger's Attack - the observable Error Condition



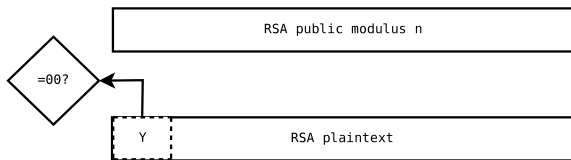
- OAEP Decoding checks that $Y = 0$
- ($Y \neq 0 \rightarrow$ “supernumerary octet”)
- $Y \neq 0$ can be learned either through
 - a specific error message
 - a shorter time to the error message compared to later OAEP errors

Manger's Attack - the observable Error Condition



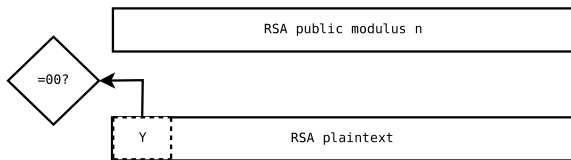
- OAEP Decoding checks that $Y = 0$
- ($Y \neq 0 \rightarrow$ “supernumerary octet”)
- $Y \neq 0$ can be learned either through
 - a specific error message
 - shorter time to the error message compared to later OAEP errors

Manger's Attack - the observable Error Condition



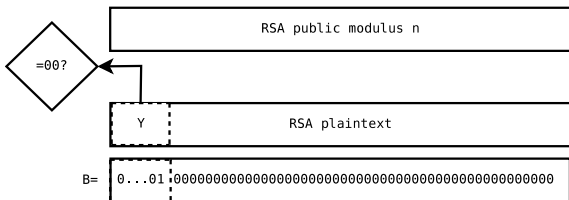
- OAEP Decoding checks that $Y = 0$
- ($Y \neq 0 \rightarrow$ “supernumerary octet”)
- $Y \neq 0$ can be learned either through
 - a specific error message
 - shorter time to the error message compared to later OAEP errors

Manger's Attack - the observable Error Condition



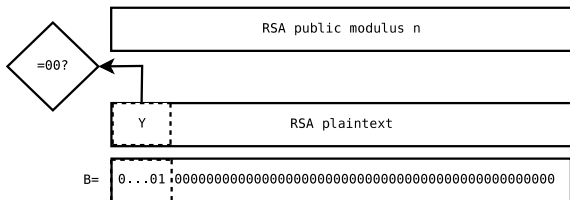
- OAEP Decoding checks that $Y = 0$
- ($Y \neq 0 \rightarrow$ “supernumerary octet”)
- $Y \neq 0$ can be learned either through
 - a specific error message
 - shorter time to the error message compared to later OAEP errors

Manger's Attack - the Information Gain



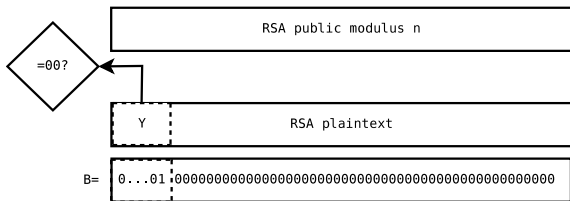
- The attacker wants to decrypt the ciphertext $c_0 = m_0^e \bmod n$
- He chooses $f \in \{0, 1, \dots, n-1\}$
- He creates ciphertexts $c_f = f^e c_0 = (fm_0)^e \bmod n$
- He observes the decryption of c_f
- If $Y \neq 0$ he learns $fm_0 \bmod n \geq B$
- Manger gives a specific strategy how to choose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

Manger's Attack - the Information Gain



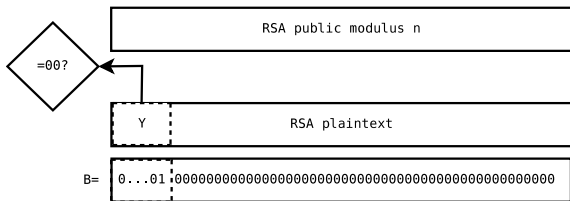
- The attacker wants to decrypt the ciphertext $c_0 = m_0^e \bmod n$
- He chooses $f \in \{0, 1, \dots, n-1\}$
- He creates ciphertexts $c_f = f^e c_0 = (fm_0)^e \bmod n$
- He observes the decryption of c_f
- If $Y \neq 0$ he learns $fm_0 \bmod n \geq B$
- Manger gives a specific strategy how to choose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

Manger's Attack - the Information Gain



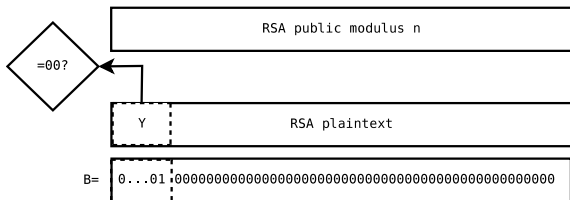
- The attacker wants to decrypt the ciphertext $c_0 = m_0^e \bmod n$
- He chooses $f \in \{0, 1, \dots, n-1\}$
- He creates ciphertexts $c_f = f^e c_0 = (fm_0)^e \bmod n$
- He observes the decryption of c_f
- If $Y \neq 0$ he learns $fm_0 \bmod n \geq B$
- Manger gives a specific strategy how to choose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

Manger's Attack - the Information Gain



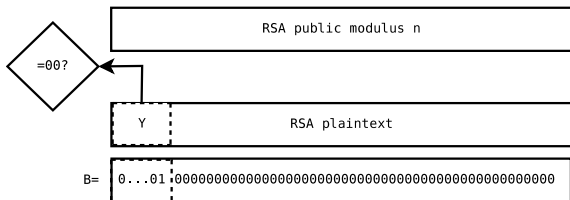
- The attacker wants to decrypt the ciphertext $c_0 = m_0^e \bmod n$
- He chooses $f \in \{0, 1, \dots, n-1\}$
- He creates ciphertexts $c_f = f^e c_0 = (fm_0)^e \bmod n$
- He observes the decryption of c_f
- If $Y \neq 0$ he learns $fm_0 \bmod n \geq B$
- Manger gives a specific strategy how to choose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

Manger's Attack - the Information Gain



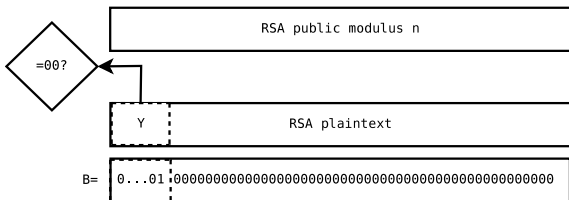
- The attacker wants to decrypt the ciphertext $c_0 = m_0^e \bmod n$
- He chooses $f \in \{0, 1, \dots, n-1\}$
- He creates ciphertexts $c_f = f^e c_0 = (fm_0)^e \bmod n$
- He observes the decryption of c_f
- If $Y \neq 0$ he learns $fm_0 \bmod n \geq B$
- Manger gives a specific strategy how to choose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

Recent Work: a 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

Recent Work: a 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

Recent Work: a 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

Potential Vulnerabilities in the Multiprecision Integer Routines

- it was also shown that in special cases there are potential vulnerabilities already in the last multiprecision integer (MPI) routine dealing with the message representative
- based on
 - counting leading zero words in the MPI routines
 - and copying of the significant words in memory
- → on 8-bit architectures: running time depends on the number of octets of m

Potential Vulnerabilities in the Multiprecision Integer Routines

- it was also shown that in special cases there are potential vulnerabilities already in the last multiprecision integer (MPI) routine dealing with the message representative
- based on
 - counting leading zero words in the MPI routines
 - and copying of the significant words in memory
- → on 8-bit architectures: running time depends on the number of octets of m

Potential Vulnerabilities in the Multiprecision Integer Routines

- it was also shown that in special cases there are potential vulnerabilities already in the last multiprecision integer (MPI) routine dealing with the message representative
- based on
 - counting leading zero words in the MPI routines
 - and copying of the significant words in memory
- → on 8-bit architectures: running time depends on the number of octets of m

Potential Vulnerabilities in the Multiprecision Integer Routines

- it was also shown that in special cases there are potential vulnerabilities already in the last multiprecision integer (MPI) routine dealing with the message representative
- based on
 - counting leading zero words in the MPI routines
 - and copying of the significant words in memory
- → on 8-bit architectures: running time depends on the number of octets of m

Potential Vulnerabilities in the Multiprecision Integer Routines

- it was also shown that in special cases there are potential vulnerabilities already in the last multiprecision integer (MPI) routine dealing with the message representative
- based on
 - counting leading zero words in the MPI routines
 - and copying of the significant words in memory
- → on 8-bit architectures: running time depends on the number of octets of m

- 1 Introduction
- 2 Decryption Oracle Attacks against the RSA Cryptosystem
- 3 Decryption Oracle Attacks against the McEliece Cryptosystem
- 4 Comparison Between the Attacks against McEliece and RSA
- 5 Countermeasures
- 6 Conclusion

- The McEliece Cryptosystem is a cryptosystem based on linear Error Correcting Codes (ECC)
 - Encoding in a linear ECC = Matrix multiplication (over \mathbb{F}_2):
 - $\vec{c} = \vec{v}G$
 - G is called a generator matrix of the code
 - code word \vec{c} is longer than message word \vec{v}
 - Decoding in a linear ECC = code specific decoding algorithm
 - decodes $\vec{c}' = \vec{c} \oplus \vec{e}$ with $\text{wt}(\vec{e}) \leq t$ back to \vec{v}
 - t is the error correcting capability of the specific code

- The McEliece Cryptosystem is a cryptosystem based on linear Error Correcting Codes (ECC)
 - Encoding in a linear ECC = Matrix multiplication (over \mathbb{F}_2):
 - $\vec{c} = \vec{v}G$
 - G is called a generator matrix of the code
 - code word \vec{c} is longer than message word \vec{v}
 - Decoding in a linear ECC = code specific decoding algorithm
 - decodes $\vec{c}' = \vec{c} \oplus \vec{e}$ with $\text{wt}(\vec{e}) \leq t$ back to \vec{v}
 - t is the error correcting capability of the specific code

- The McEliece Cryptosystem is a cryptosystem based on linear Error Correcting Codes (ECC)
 - Encoding in a linear ECC = Matrix multiplication (over \mathbb{F}_2):
 - $\vec{c} = \vec{v}G$
 - G is called a generator matrix of the code
 - code word \vec{c} is longer than message word \vec{v}
 - Decoding in a linear ECC = code specific decoding algorithm
 - decodes $\vec{c}' = \vec{c} \oplus \vec{e}$ with $\text{wt}(\vec{e}) \leq t$ back to \vec{v}
 - t is the error correcting capability of the specific code

- The McEliece Cryptosystem is a cryptosystem based on linear Error Correcting Codes (ECC)
 - Encoding in a linear ECC = Matrix multiplication (over \mathbb{F}_2):
 - $\vec{c} = \vec{v}G$
 - G is called a generator matrix of the code
 - code word \vec{c} is longer than message word \vec{v}
 - Decoding in a linear ECC = code specific decoding algorithm
 - decodes $\vec{c}' = \vec{c} \oplus \vec{e}$ with $\text{wt}(\vec{e}) \leq t$ back to \vec{v}
 - t is the error correcting capability of the specific code

- The McEliece Cryptosystem is a cryptosystem based on linear Error Correcting Codes (ECC)
 - Encoding in a linear ECC = Matrix multiplication (over \mathbb{F}_2):
 - $\vec{c} = \vec{v}G$
 - G is called a generator matrix of the code
 - code word \vec{c} is longer than message word \vec{v}
 - Decoding in a linear ECC = code specific decoding algorithm
 - decodes $\vec{c}' = \vec{c} \oplus \vec{e}$ with $\text{wt}(\vec{e}) \leq t$ back to \vec{v}
 - t is the error correcting capability of the specific code

- The McEliece Cryptosystem is a cryptosystem based on linear Error Correcting Codes (ECC)
 - Encoding in a linear ECC = Matrix multiplication (over \mathbb{F}_2):
 - $\vec{c} = \vec{v}G$
 - G is called a generator matrix of the code
 - code word \vec{c} is longer than message word \vec{v}
 - Decoding in a linear ECC = code specific decoding algorithm
 - decodes $\vec{c}' = \vec{c} \oplus \vec{e}$ with $\text{wt}(\vec{e}) \leq t$ back to \vec{v}
 - t is the error correcting capability of the specific code

- The McEliece Cryptosystem is a cryptosystem based on linear Error Correcting Codes (ECC)
 - Encoding in a linear ECC = Matrix multiplication (over \mathbb{F}_2):
 - $\vec{c} = \vec{v}G$
 - G is called a generator matrix of the code
 - code word \vec{c} is longer than message word \vec{v}
 - Decoding in a linear ECC = code specific decoding algorithm
 - decodes $\vec{c}' = \vec{c} \oplus \vec{e}$ with $\text{wt}(\vec{e}) \leq t$ back to \vec{v}
 - t is the error correcting capability of the specific code

- The McEliece Cryptosystem is a cryptosystem based on linear Error Correcting Codes (ECC)
 - Encoding in a linear ECC = Matrix multiplication (over \mathbb{F}_2):
 - $\vec{c} = \vec{v}G$
 - G is called a generator matrix of the code
 - code word \vec{c} is longer than message word \vec{v}
 - Decoding in a linear ECC = code specific decoding algorithm
 - decodes $\vec{c}' = \vec{c} \oplus \vec{e}$ with $\text{wt}(\vec{e}) \leq t$ back to \vec{v}
 - t is the error correcting capability of the specific code

- McEliece Encryption: \approx Encoding a message word in an unknown code:
 - $\vec{z} = \vec{v}G_p \oplus \vec{e}$, $\text{wt}(\vec{e}) = t$
 - G_p is the public key (public generator matrix)
- McEliece Decryption: possible because the secret code is known:
 - $G_p \neq G_s$
 - but the actual secret code is "hidden" in G_p

- McEliece Encryption: \approx Encoding a message word in an unknown code:
 - $\vec{z} = \vec{v}G_p \oplus \vec{e}$, $\text{wt}(\vec{e}) = t$
 - G_p is the public key (public generator matrix)
- McEliece Decryption: possible because the secret code is known:
 - $G_p \neq G_s$
 - but the actual secret code is "hidden" in G_p

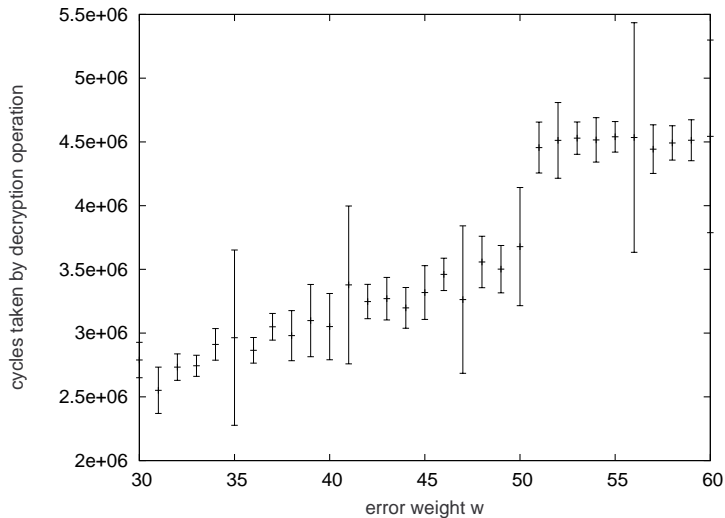
- McEliece Encryption: \approx Encoding a message word in an unknown code:
 - $\vec{z} = \vec{v}G_p \oplus \vec{e}$, $\text{wt}(\vec{e}) = t$
 - G_p is the public key (public generator matrix)
- McEliece Decryption: possible because the secret code is known:
 - $G_p \neq G_s$
 - but the actual secret code is "hidden" in G_p

- McEliece Encryption: \approx Encoding a message word in an unknown code:
 - $\vec{z} = \vec{v}G_p \oplus \vec{e}$, $\text{wt}(\vec{e}) = t$
 - G_p is the public key (public generator matrix)
- McEliece Decryption: possible because the secret code is known:
 - $G_p \neq G_s$
 - but the actual secret code is “hidden” in G_p

- McEliece Encryption: \approx Encoding a message word in an unknown code:
 - $\vec{z} = \vec{v}G_p \oplus \vec{e}$, $\text{wt}(\vec{e}) = t$
 - G_p is the public key (public generator matrix)
- McEliece Decryption: possible because the secret code is known:
 - $G_p \neq G_s$
 - but the actual secret code is “hidden” in G_p

- McEliece Encryption: \approx Encoding a message word in an unknown code:
 - $\vec{z} = \vec{v}G_p \oplus \vec{e}$, $\text{wt}(\vec{e}) = t$
 - G_p is the public key (public generator matrix)
- McEliece Decryption: possible because the secret code is known:
 - $G_p \neq G_s$
 - but the actual secret code is “hidden” in G_p

Timing Effects in the McEliece Decryption for $t = 50$



Exploitation of the Timing Effects

- An attacker wishes to decrypt a certain ciphertext \vec{z}
- he creates manipulated versions of this ciphertext:
 - in each he flips a different bit
 - and thus carries now $t - 1$ or $t + 1$ errors
- he observes the decryption and (through timing) tries to determine whether
 - $wt(\vec{e}) = t - 1 \rightarrow$ the flipped bit was an error position
 - $wt(\vec{e}) = t + 1 \rightarrow$ the flipped bit was NOT an error position
- he reconstructs \vec{e} used during encryption and thus can recover the message

Exploitation of the Timing Effects

- An attacker wishes to decrypt a certain ciphertext \vec{z}
- he creates manipulated versions of this ciphertext:
 - in each he flips a different bit
 - and thus carries now $t - 1$ or $t + 1$ errors
- he observes the decryption and (through timing) tries to determine whether
 - $wt(\vec{e}) = t - 1 \rightarrow$ the flipped bit was an error position
 - $wt(\vec{e}) = t + 1 \rightarrow$ the flipped bit was NOT an error position
- he reconstructs \vec{e} used during encryption and thus can recover the message

Exploitation of the Timing Effects

- An attacker wishes to decrypt a certain ciphertext \vec{z}
- he creates manipulated versions of this ciphertext:
 - in each he flips a different bit
 - and thus carries now $t - 1$ or $t + 1$ errors
- he observes the decryption and (through timing) tries to determine whether
 - $\text{wt}(\vec{e}) = t - 1 \rightarrow$ the flipped bit was an error position
 - $\text{wt}(\vec{e}) = t + 1 \rightarrow$ the flipped bit was NOT an error position
- he reconstructs \vec{e} used during encryption and thus can recover the message

Exploitation of the Timing Effects

- An attacker wishes to decrypt a certain ciphertext \vec{z}
- he creates manipulated versions of this ciphertext:
 - in each he flips a different bit
 - and thus carries now $t - 1$ or $t + 1$ errors
- he observes the decryption and (through timing) tries to determine whether
 - $wt(\vec{e}) = t - 1 \rightarrow$ the flipped bit was an error position
 - $wt(\vec{e}) = t + 1 \rightarrow$ the flipped bit was NOT an error position
- he reconstructs \vec{e} used during encryption and thus can recover the message

Exploitation of the Timing Effects

- An attacker wishes to decrypt a certain ciphertext \vec{z}
- he creates manipulated versions of this ciphertext:
 - in each he flips a different bit
 - and thus carries now $t - 1$ or $t + 1$ errors
- he observes the decryption and (through timing) tries to determine whether
 - $\text{wt}(\vec{e}) = t - 1 \rightarrow$ the flipped bit was an error position
 - $\text{wt}(\vec{e}) = t + 1 \rightarrow$ the flipped bit was NOT an error position
- he reconstructs \vec{e} used during encryption and thus can recover the message

Exploitation of the Timing Effects

- An attacker wishes to decrypt a certain ciphertext \vec{z}
- he creates manipulated versions of this ciphertext:
 - in each he flips a different bit
 - and thus carries now $t - 1$ or $t + 1$ errors
- he observes the decryption and (through timing) tries to determine whether
 - $\text{wt}(\vec{e}) = t - 1 \rightarrow$ the flipped bit was an error position
 - $\text{wt}(\vec{e}) = t + 1 \rightarrow$ the flipped bit was NOT an error position
- he reconstructs \vec{e} used during encryption and thus can recover the message

Exploitation of the Timing Effects

- An attacker wishes to decrypt a certain ciphertext \vec{z}
- he creates manipulated versions of this ciphertext:
 - in each he flips a different bit
 - and thus carries now $t - 1$ or $t + 1$ errors
- he observes the decryption and (through timing) tries to determine whether
 - $\text{wt}(\vec{e}) = t - 1 \rightarrow$ the flipped bit was an error position
 - $\text{wt}(\vec{e}) = t + 1 \rightarrow$ the flipped bit was NOT an error position
- he reconstructs \vec{e} used during encryption and thus can recover the message

Exploitation of the Timing Effects

- An attacker wishes to decrypt a certain ciphertext \vec{z}
- he creates manipulated versions of this ciphertext:
 - in each he flips a different bit
 - and thus carries now $t - 1$ or $t + 1$ errors
- he observes the decryption and (through timing) tries to determine whether
 - $\text{wt}(\vec{e}) = t - 1 \rightarrow$ the flipped bit was an error position
 - $\text{wt}(\vec{e}) = t + 1 \rightarrow$ the flipped bit was NOT an error position
- he reconstructs \vec{e} used during encryption and thus can recover the message

A new Vulnerability in the Root Finding Algorithm

- During the McEliece decryption the Error Locator Polynomial (ELP) plays a key role
 - it is computed
 - its roots (zeros) are determined
- Previous works on timing attacks took only into account the effect of the degree of the ELP on the running time (\rightarrow linear incline for $\text{wt}(\vec{e}) \leq t$)
- But an efficient root finding algorithm can introduce a new vulnerability:
 - to speed up the time consuming root finding \rightarrow factoring of the ELP
 - but: the number of roots resp. factor polynomials in the ELP is different for
 - $\text{wt}(\vec{e}) \leq t$
 - $\text{wt}(\vec{e}) > t$
 - \rightarrow difference in running time for these two cases!

A new Vulnerability in the Root Finding Algorithm

- During the McEliece decryption the Error Locator Polynomial (ELP) plays a key role
 - it is computed
 - its roots (zeros) are determined
- Previous works on timing attacks took only into account the effect of the degree of the ELP on the running time (\rightarrow linear incline for $\text{wt}(\vec{e}) \leq t$)
- But an efficient root finding algorithm can introduce a new vulnerability:
 - to speed up the time consuming root finding \rightarrow factoring of the ELP
 - but: the number of roots resp. factor polynomials in the ELP is different for
 - $\text{wt}(\vec{e}) \leq t$
 - $\text{wt}(\vec{e}) > t$
 - \rightarrow difference in running time for these two cases!

A new Vulnerability in the Root Finding Algorithm

- During the McEliece decryption the Error Locator Polynomial (ELP) plays a key role
 - it is computed
 - its roots (zeros) are determined
- Previous works on timing attacks took only into account the effect of the degree of the ELP on the running time (\rightarrow linear incline for $\text{wt}(\vec{e}) \leq t$)
- But an efficient root finding algorithm can introduce a new vulnerability:
 - to speed up the time consuming root finding \rightarrow factoring of the ELP
 - but: the number of roots resp. factor polynomials in the ELP is different for
 - $\text{wt}(\vec{e}) \leq t$
 - $\text{wt}(\vec{e}) > t$
 - \rightarrow difference in running time for these two cases!

A new Vulnerability in the Root Finding Algorithm

- During the McEliece decryption the Error Locator Polynomial (ELP) plays a key role
 - it is computed
 - its roots (zeros) are determined
- Previous works on timing attacks took only into account the effect of the degree of the ELP on the running time (\rightarrow linear incline for $\text{wt}(\vec{e}) \leq t$)
- But an efficient root finding algorithm can introduce a new vulnerability:
 - to speed up the time consuming root finding \rightarrow factoring of the ELP
 - but: the number of roots resp. factor polynomials in the ELP is different for
 - $\text{wt}(\vec{e}) \leq t$
 - $\text{wt}(\vec{e}) > t$
 - \rightarrow difference in running time for these two cases!

A new Vulnerability in the Root Finding Algorithm

- During the McEliece decryption the Error Locator Polynomial (ELP) plays a key role
 - it is computed
 - its roots (zeros) are determined
- Previous works on timing attacks took only into account the effect of the degree of the ELP on the running time (\rightarrow linear incline for $\text{wt}(\vec{e}) \leq t$)
- But an efficient root finding algorithm can introduce a new vulnerability:
 - to speed up the time consuming root finding \rightarrow factoring of the ELP
 - but: the number of roots resp. factor polynomials in the ELP is different for
 - $\text{wt}(\vec{e}) \leq t$
 - $\text{wt}(\vec{e}) > t$
 - \rightarrow difference in running time for these two cases!

A new Vulnerability in the Root Finding Algorithm

- During the McEliece decryption the Error Locator Polynomial (ELP) plays a key role
 - it is computed
 - its roots (zeros) are determined
- Previous works on timing attacks took only into account the effect of the degree of the ELP on the running time (\rightarrow linear incline for $\text{wt}(\vec{e}) \leq t$)
- But an efficient root finding algorithm can introduce a new vulnerability:
 - to speed up the time consuming root finding \rightarrow factoring of the ELP
 - but: the number of roots resp. factor polynomials in the ELP is different for
 - $\text{wt}(\vec{e}) \leq t$
 - $\text{wt}(\vec{e}) > t$
 - \rightarrow difference in running time for these two cases!

A new Vulnerability in the Root Finding Algorithm

- During the McEliece decryption the Error Locator Polynomial (ELP) plays a key role
 - it is computed
 - its roots (zeros) are determined
- Previous works on timing attacks took only into account the effect of the degree of the ELP on the running time (\rightarrow linear incline for $\text{wt}(\vec{e}) \leq t$)
- But an efficient root finding algorithm can introduce a new vulnerability:
 - to speed up the time consuming root finding \rightarrow factoring of the ELP
 - but: the number of roots resp. factor polynomials in the ELP is different for
 - $\text{wt}(\vec{e}) \leq t$
 - $\text{wt}(\vec{e}) > t$
 - \rightarrow difference in running time for these two cases!

A new Vulnerability in the Root Finding Algorithm

- During the McEliece decryption the Error Locator Polynomial (ELP) plays a key role
 - it is computed
 - its roots (zeros) are determined
- Previous works on timing attacks took only into account the effect of the degree of the ELP on the running time (\rightarrow linear incline for $\text{wt}(\vec{e}) \leq t$)
- But an efficient root finding algorithm can introduce a new vulnerability:
 - to speed up the time consuming root finding \rightarrow factoring of the ELP
 - but: the number of roots resp. factor polynomials in the ELP is different for
 - $\text{wt}(\vec{e}) \leq t$
 - $\text{wt}(\vec{e}) > t$
 - \rightarrow difference in running time for these two cases!

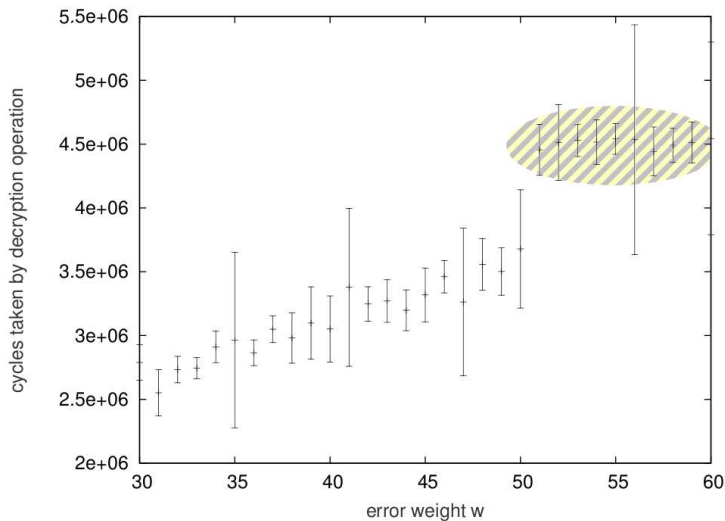
A new Vulnerability in the Root Finding Algorithm

- During the McEliece decryption the Error Locator Polynomial (ELP) plays a key role
 - it is computed
 - its roots (zeros) are determined
- Previous works on timing attacks took only into account the effect of the degree of the ELP on the running time (\rightarrow linear incline for $\text{wt}(\vec{e}) \leq t$)
- But an efficient root finding algorithm can introduce a new vulnerability:
 - to speed up the time consuming root finding \rightarrow factoring of the ELP
 - but: the number of roots resp. factor polynomials in the ELP is different for
 - $\text{wt}(\vec{e}) \leq t$
 - $\text{wt}(\vec{e}) > t$
 - \rightarrow difference in running time for these two cases!

A new Vulnerability in the Root Finding Algorithm

- During the McEliece decryption the Error Locator Polynomial (ELP) plays a key role
 - it is computed
 - its roots (zeros) are determined
- Previous works on timing attacks took only into account the effect of the degree of the ELP on the running time (\rightarrow linear incline for $\text{wt}(\vec{e}) \leq t$)
- But an efficient root finding algorithm can introduce a new vulnerability:
 - to speed up the time consuming root finding \rightarrow factoring of the ELP
 - but: the number of roots resp. factor polynomials in the ELP is different for
 - $\text{wt}(\vec{e}) \leq t$
 - $\text{wt}(\vec{e}) > t$
 - \rightarrow difference in running time for these two cases!

Timing Effects from the Factoring inside the Root Finding Algorithm



- 1 Introduction
- 2 Decryption Oracle Attacks against the RSA Cryptosystem
- 3 Decryption Oracle Attacks against the McEliece Cryptosystem
- 4 Comparison Between the Attacks against McEliece and RSA
- 5 Countermeasures
- 6 Conclusion

Comparison of the McEliece and RSA cryptosystems

	RSA	McEliece
homom. Property	$\mathcal{E}(a) \cdot \mathcal{E}(b) \equiv \mathcal{E}(a \cdot b) \pmod{n}$	$\mathcal{E}(a) \oplus \mathcal{E}(b) = \mathcal{E}(a \oplus b)$
observ. Prop.	(lead. octet = 0?) \neq octets in m	wt(\vec{e})
Decryption
	Final \mathbb{Z}_n Operation	comp. ELP Root Finding for ELP
Message Encoding	Encoding in \mathbb{Z}_n	Encoding in \mathbb{F}_2^n
CCA2 Check	OAEP Check	appropriate CCA2 Check

- 1 Introduction
- 2 Decryption Oracle Attacks against the RSA Cryptosystem
- 3 Decryption Oracle Attacks against the McEliece Cryptosystem
- 4 Comparison Between the Attacks against McEliece and RSA
- 5 Countermeasures
- 6 Conclusion

Ideal Countermeasures

	RSA	McEliece
homom. Property	$\mathcal{E}(a) \cdot \mathcal{E}(b) \equiv \mathcal{E}(a \cdot b) \pmod{n}$	$\mathcal{E}(a) \oplus \mathcal{E}(b) = \mathcal{E}(a \oplus b)$
observ. Prop.	(lead. octet = 0?) \neq octets in m	wt(\vec{e})
Decryption
	Final \mathbb{Z}_n Operation	comp. ELP Root Finding for ELP
Message Encoding	Encoding in \mathbb{Z}_n	Encoding in \mathbb{F}_2^n
CCA2 Check	OAEP Check	appropriate CCA2 Check

- Ideal Countermeasures would already ensure the *observable plaintext property* to be unambiguous during the basic decryption
- then the subsequent operations (encoding of the message representative and the CCA2-conversion) would be relieved from countermeasures
- In McEliece the number of errors can be forced to be t during decryption
- But RSA generally allows any number of leading zero octets !

- Ideal Countermeasures would already ensure the *observable plaintext property* to be unambiguous during the basic decryption
- then the subsequent operations (encoding of the message representative and the CCA2-conversion) would be relieved from countermeasures
- In McEliece the number of errors can be forced to be t during decryption
- But RSA generally allows any number of leading zero octets !

- Ideal Countermeasures would already ensure the *observable plaintext property* to be unambiguous during the basic decryption
- then the subsequent operations (encoding of the message representative and the CCA2-conversion) would be relieved from countermeasures
- In McEliece the number of errors can be forced to be t during decryption
- But RSA generally allows any number of leading zero octets !

- Ideal Countermeasures would already ensure the *observable plaintext property* to be unambiguous during the basic decryption
- then the subsequent operations (encoding of the message representative and the CCA2-conversion) would be relieved from countermeasures
- In McEliece the number of errors can be forced to be t during decryption
- But RSA generally allows any number of leading zero octets !

- the critical plaintext property

- (RSA: number of leading zero octets)
- McEliece: number of "errors" in ciphertext)

must not be revealed through timing

- to this end

- certain algorithm part must have timing irrespective of that plaintext property (e.g. encoding of \mathbb{Z}_n elements)
- at certain points irregular data simply should be ignored (e.g. non-zero value of the "leading octet" Y in RSA-OAEP)
- at certain points fake data has to be created (McEliece)
not truly randomly!

but *not* pseudorandomly derived from the ciphertexts
else the indeterministic behaviour of the decryption oracle might indicate the critical plaintext property

- While usage of the actual key can be avoided, the plaintext will always appear in the computation

- the critical plaintext property
 - (RSA: number of leading zero octets)
 - McEliece: number of "errors" in ciphertext)

must not be revealed through timing

- to this end
 - certain algorithm part must have timing irrespective of that plaintext property (e.g. encoding of \mathbb{Z}_n elements)
 - at certain points irregular data simply should be ignored (e.g. non-zero value of the "leading octet" Y in RSA-OAEP)
 - at certain points fake data has to be created (McEliece)
not truly randomly!
 - but pseudo-randomly derived from the ciphertext
 - else the indeterministic behaviour of the decryption oracle might indicate the critical plaintext property
- While usage of the actual key can be avoided, the plaintext will always appear in the computation

- the critical plaintext property
 - (RSA: number of leading zero octets)
 - McEliece: number of “errors” in ciphertext)

must not be revealed through timing

- to this end

- certain algorithm part must have timing irrespective of that plaintext property (e.g. encoding of \mathbb{Z}_n elements)
- at certain points irregular data simply should be ignored (e.g. non-zero value of the “leading octet” Y in RSA-OAEP)
- at certain points fake data has to be created (McEliece)
not truly randomly!

but *permanently* derived from the ciphertexts
else the indeterministic behaviour of the decryption oracle might indicate the critical plaintext property

- While usage of the actual key can be avoided, the plaintext will always appear in the computation

- the critical plaintext property
 - (RSA: number of leading zero octets)
 - McEliece: number of “errors” in ciphertext)must not be revealed through timing
- to this end
 - certain algorithm part must have timing irrespective of that plaintext property (e.g. encoding of \mathbb{Z}_n elements)
 - at certain points irregular data simply should be ignored (e.g. non-zero value of the “leading octet” Y in RSA-OAEP)
 - at certain points fake data has to be created (McEliece)
not truly randomly!
 - but pseudorandomly derived from the ciphertext
 - else the indeterministic behaviour of the decryption oracle might indicate the critical plaintext property
- While usage of the actual key can be avoided, the plaintext will always appear in the computation

- the critical plaintext property
 - (RSA: number of leading zero octets)
 - McEliece: number of “errors” in ciphertext)must not be revealed through timing
- to this end
 - certain algorithm part must have timing irrespective of that plaintext property (e.g. encoding of \mathbb{Z}_n elements)
 - at certain points irregular data simply should be ignored (e.g. non-zero value of the “leading octet” Y in RSA-OAEP)
 - at certain points fake data has to be created (McEliece)
not truly randomly!
 - but pseudorandomly derived from the ciphertext
 - else the indeterministic behaviour of the decryption oracle might indicate the critical plaintext property
- While usage of the actual key can be avoided, the plaintext will always appear in the computation

- the critical plaintext property
 - (RSA: number of leading zero octets)
 - McEliece: number of “errors” in ciphertext)must not be revealed through timing
- to this end
 - certain algorithm part must have timing irrespective of that plaintext property (e.g. encoding of \mathbb{Z}_n elements)
 - at certain points irregular data simply should be ignored (e.g. non-zero value of the “leading octet” Y in RSA-OAEP)
 - at certain points fake data has to be created (McEliece)
not truly randomly!
 - but pseudorandomly derived from the ciphertext
 - else the indeterministic behaviour of the decryption oracle might indicate the critical plaintext property
- While usage of the actual key can be avoided, the plaintext will always appear in the computation

- the critical plaintext property
 - (RSA: number of leading zero octets)
 - McEliece: number of “errors” in ciphertext)must not be revealed through timing
- to this end
 - certain algorithm part must have timing irrespective of that plaintext property (e.g. encoding of \mathbb{Z}_n elements)
 - at certain points irregular data simply should be ignored (e.g. non-zero value of the “leading octet” Y in RSA-OAEP)
 - at certain points fake data has to be created (McEliece)
not truly randomly!
 - but pseudorandomly derived from the ciphertextelse the indeterministic behaviour of the decryption oracle might indicate the critical plaintext property
- While usage of the actual key can be avoided, the plaintext will always appear in the computation

- the critical plaintext property
 - (RSA: number of leading zero octets)
 - McEliece: number of “errors” in ciphertext)must not be revealed through timing
- to this end
 - certain algorithm part must have timing irrespective of that plaintext property (e.g. encoding of \mathbb{Z}_n elements)
 - at certain points irregular data simply should be ignored (e.g. non-zero value of the “leading octet” Y in RSA-OAEP)
 - at certain points fake data has to be created (McEliece)
not truly randomly!
 - but pseudorandomly derived from the ciphertextelse the indeterministic behaviour of the decryption oracle might indicate the critical plaintext property
- While usage of the actual key can be avoided, the plaintext will always appear in the computation

- the critical plaintext property
 - (RSA: number of leading zero octets)
 - McEliece: number of “errors” in ciphertext)must not be revealed through timing
- to this end
 - certain algorithm part must have timing irrespective of that plaintext property (e.g. encoding of \mathbb{Z}_n elements)
 - at certain points irregular data simply should be ignored (e.g. non-zero value of the “leading octet” Y in RSA-OAEP)
 - at certain points fake data has to be created (McEliece)
not truly randomly!
 - but pseudorandomly derived from the ciphertextelse the indeterministic behaviour of the decryption oracle might indicate the critical plaintext property
- While usage of the actual key can be avoided, the plaintext will always appear in the computation

- 1 Introduction
- 2 Decryption Oracle Attacks against the RSA Cryptosystem
- 3 Decryption Oracle Attacks against the McEliece Cryptosystem
- 4 Comparison Between the Attacks against McEliece and RSA
- 5 Countermeasures
- 6 Conclusion

- With respect to message aimed side channel attacks,
 - We showed recent results for the RSA cryptosystem and new results for the McEliece cryptosystem
 - By structuring and comparing the vulnerabilities of both cryptosystems, we outlined the general approach for the analysis of *public key cryptosystems with homomorphic properties*
 - We pointed some aspects concerning the countermeasures against such attacks

- With respect to message aimed side channel attacks,
 - We showed recent results for the RSA cryptosystem and new results for the McEliece cryptosystem
 - By structuring and comparing the vulnerabilities of both cryptosystems, we outlined the general approach for the analysis of *public key cryptosystems with homomorphic properties*
 - We pointed some aspects concerning the countermeasures against such attacks

- With respect to message aimed side channel attacks,
 - We showed recent results for the RSA cryptosystem and new results for the McEliece cryptosystem
 - By structuring and comparing the vulnerabilities of both cryptosystems, we outlined the general approach for the analysis of *public key cryptosystems with homomorphic properties*
 - We pointed some aspects concerning the countermeasures against such attacks

- With respect to message aimed side channel attacks,
 - We showed recent results for the RSA cryptosystem and new results for the McEliece cryptosystem
 - By structuring and comparing the vulnerabilities of both cryptosystems, we outlined the general approach for the analysis of *public key cryptosystems with homomorphic properties*
 - We pointed some aspects concerning the countermeasures against such attacks

- Thank You!