Addition with Blinded Operands

Mohamed Karroumi[*] ▪ Benjamin Richard ▪ Marc Joye

technicolor

Addition with Blinded Operands

Mohamed Karroumi* ▪ Benjamin Richard ▪ Marc Joye

technicolor

Outline

1. **Preliminary Background**
   - DPA attacks and countermeasures
   - Masking and switching method
2. **A new DPA resistant addition algorithm**
   - Basic algorithm
   - DPA resistant addition algorithm
3. **Application to XTEA**
   - XTEA overview
   - Preventing first-order DPA
   - Performance analysis
4. **Conclusion**

technicolor

Outline

technicolor

Differential Power Analysis

- Side channel attack
- DPA introduced by Paul Kocher et al. 1998
- Recovers secret keys used for en/decryption
  - Some a priori knowledge of the algorithm is required
- Power consumption depends on data being processed
  - Power measurements give hints about processed internal data
- When key cannot be found directly in a single power trace
  - Gather many power consumption curves
  - Assume a part of the key value, divide data into two groups(0 and 1 for chosen bit), calculate mean value curve of each group
  - Correlation between predicted power consumption and actual power consumption
  - If the subkey guess is correct, then the prediction (likely) matches the physical measurement

Differential Power Analysis

- Side channel attack
- DPA introduced by Paul Kocher et al. 1998
- Recovers secret keys used for en/decryption
  - Some a priori knowledge of the algorithm is required
- Power consumption depends on data being processed
  - Power measurements give hints about processed internal data
- When key cannot be found directly in a single power trace
  - Gather many power consumption curves
  - Assume a part of the key value, divide data into two groups(0 and 1 for chosen bit), calculate mean value curve of each group
  - Correlation between predicted power consumption and actual power consumption
  - If the subkey guess is correct, then the prediction (likely) matches the physical measurement

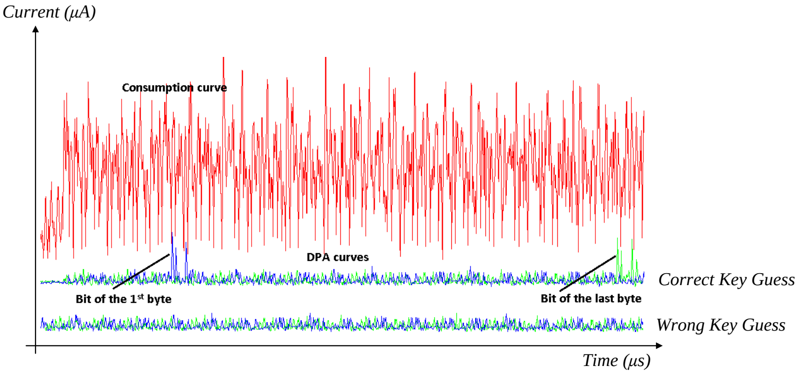
Differential Power Analysis

- Side channel attack
- DPA introduced by Paul Kocher et al. 1998
- Recovers secret keys used for en/decryption
  - Some a priori knowledge of the algorithm is required
- Power consumption depends on data being processed
  - Power measurements give hints about processed internal data
- When key cannot be found directly in a single power trace
  - Gather many power consumption curves
  - Assume a part of the key value, divide data into two groups(0 and 1 for chosen bit), calculate mean value curve of each group
  - Correlation between predicted power consumption and actual power consumption
  - If the subkey guess is correct, then the prediction (likely) matches the physical measurement

technicolor

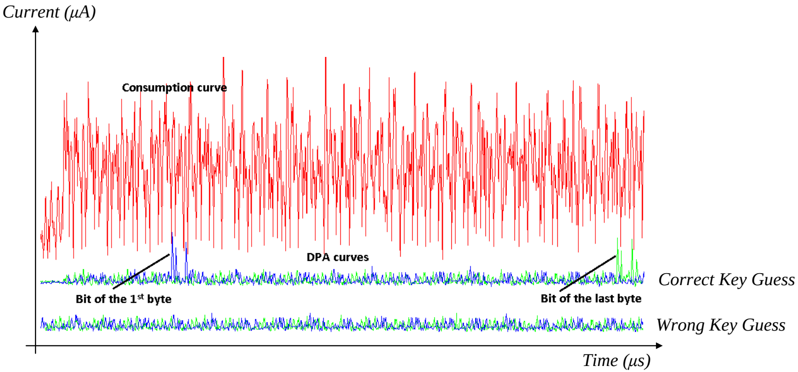
DPA results example

- DPA and power curves superposition
- Correct subkey predicted ⇒ spikes in the differential curves
- Repeat the process for other parts of the key
- Exhaustive search for remaining bits of the key



DPA results example

- DPA and power curves superposition
- Correct subkey predicted ⇒ spikes in the differential curves
- Repeat the process for other parts of the key
- Exhaustive search for remaining bits of the key



A DPA countermeasure

- An approach is to randomize the intermediate results
  - the power consumption of the device processing randomized data is not correlated to the intermediate results

- Masking: can be applied in software or hardware
  - Split intermediate variables into at least two shares during execution (Chari et al. 1999)
  - Power leakage of one share does not leak sensitive information
  - Two shares (a random mask and masked variable) are sufficient to protect against first-order DPA

- Two common masking techniques
  - Boolean masking: $x \to (X = x \oplus r_x, r_x)$
  - Arithmetic masking: $x \to (X = x - r_x, r_x)$

technicolor

A DPA countermeasure

- An approach is to randomize the intermediate results
  - the power consumption of the device processing randomized data is not correlated to the intermediate results
- Masking: can be applied in software or hardware
  - Split intermediate variables into at least two shares during execution (Chari et al. 1999)
  - Power leakage of one share does not leak sensitive information
  - Two shares (a random mask and masked variable) are sufficient to protect against first-order DPA
- Two common masking techniques
  - Boolean masking: $x \to (\mathbf{X} = x \oplus r_x, r_x)$
  - Arithmetic masking: $x \to (\mathbf{X} = x - r_x, r_x)$

technicolor

A DPA countermeasure

- An approach is to randomize the intermediate results
  - the power consumption of the device processing randomized data is not correlated to the intermediate results

- Masking: can be applied in software or hardware
  - Split intermediate variables into at least two shares during execution (Chari et al. 1999)
  - Power leakage of one share does not leak sensitive information
  - Two shares (a random mask and masked variable) are sufficient to protect against first-order DPA

- Two common masking techniques
  - Boolean masking: $x \to (\mathbf{X} = x \oplus r_x, r_x)$
  - Arithmetic masking: $x \to (\mathbf{X} = x - r_x, r_x)$

  $\Rightarrow$ For algorithms that combine both types of operations, a secure conversion from one masking to another must be used (Messerges 2000)

technicolor

A DPA countermeasure

- An approach is to randomize the intermediate results
  - the power consumption of the device processing randomized data is not correlated to the intermediate results
- Masking: can be applied in software or hardware
  - Split intermediate variables into at least two shares during execution (Chari et al. 1999)
  - Power leakage of one share does not leak sensitive information
  - Two shares (a random mask and masked variable) are sufficient to protect against first-order DPA
- Two common masking techniques
  - Boolean masking: $x \to (\mathbf{X} = x \oplus r_x, r_x)$
  - Arithmetic masking: $x \to (\mathbf{X} = x - r_x, r_x)$

  $\Rightarrow$ For algorithms that combine both types of operations, a secure conversion from one masking to another must be used (Messerges 2000)

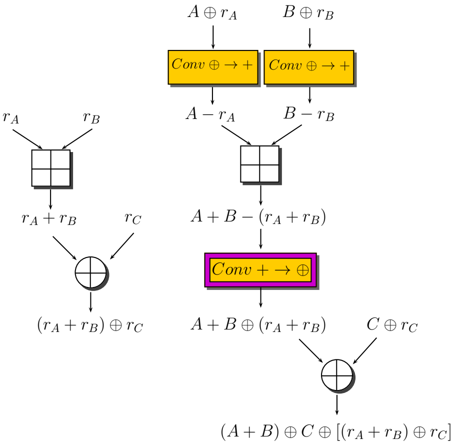
technicolor

Mask-switching methods

Example

- Securely compute $(A + B) \oplus C$ with boolean masked variables

- 2 B-to-A and 1 A-to-B conversions needed
- B-to-A is efficient and costs 7 ops (Goubin 2001)
- A-to-B is less efficient and costs $5k + 5$ ops (Goubin)

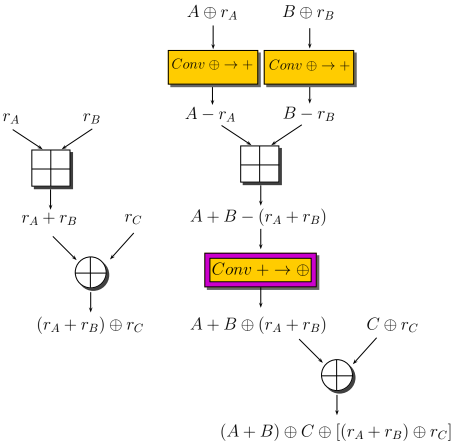


Mask-switching methods

Example

- Securely compute $(A + B) \oplus C$ with boolean masked variables

- 2 B-to-A and 1 A-to-B conversions needed
- B-to-A is efficient and costs 7 ops (Goubin 2001)
- A-to-B is less efficient and costs $5k + 5$ ops (Goubin)



technicolor

Mask-switching with LUTs

- In 2003, Coron and Tchulkine propose to use pre-computed tables to perform A-to-B conversion
  - A table G is used to convert nibbles (i.e. 4 or 8-bit part of the variables) from arithmetic to Boolean masking
  - The input of the table G is masked (additively) and viewed during conversion step as a memory offset information
  - The table offset contains the corresponding (Boolean) masked variable

- The method was later improved by Neiße and Pulkus in 2004
  - Reduces the RAM consumption

- An extension to the above techniques was more recently proposed by Debraize in 2012
  - Offers better security
  - Interesting for 8-bit CPUs

Mask-switching with LUTs

- In 2003, Coron and Tchulkine propose to use pre-computed tables to perform A-to-B conversion
  - A table *G* is used to convert nibbles (i.e. 4 or 8-bit part of the variables) from arithmetic to Boolean masking
  - The input of the table *G* is masked (additively) and viewed during conversion step as a memory offset information
  - The table offset contains the corresponding (Boolean) masked variable

- The method was later improved by Neiβe and Pulkus in 2004
  - Reduces the RAM consumption

- An extension to the above techniques was more recently proposed by Debraize in 2012
  - Offers better security
  - Interesting for 8-bit CPUs

Mask-switching with LUTs

- In 2003, Coron and Tchulkine propose to use pre-computed tables to perform A-to-B conversion
  - A table *G* is used to convert nibbles (i.e. 4 or 8-bit part of the variables) from arithmetic to Boolean masking
  - The input of the table *G* is masked (additively) and viewed during conversion step as a memory offset information
  - The table offset contains the corresponding (Boolean) masked variable

- The method was later improved by Neiβe and Pulkus in 2004
  - Reduces the RAM consumption

- An extension to the above techniques was more recently proposed by Debraize in 2012
  - Offers better security
  - Interesting for 8-bit CPUs

technicolor

This Talk

Mask-switching method

$$
\begin{array}{ccccc}
x \oplus r_x & & y \oplus r_y & & \mathbf{s} = (x+y) \oplus (r_x + r_y) \\
\Big\Downarrow \text{ Secure B-to-A} & & \Big\Downarrow & & \Big\Uparrow \text{ Secure A-to-B (with LUTs)} \\
x - r_x & & y - r_y & \xrightarrow[\text{(classical)}]{+} & (x+y) - (r_x + r_y)
\end{array}
$$

- If we have only one addition (followed by boolean operations) can we avoid mask-switching ?

New method

- The new proposed algorithm is based on a more direct approacxh

$$
x \oplus r_x \qquad y \oplus r_y \qquad \xRightarrow{\text{Secure adder}} \qquad \mathbf{s} = (x+y) \oplus (r_x \oplus r_y)
$$

This Talk

Mask-switching method

$$
\begin{array}{ccccc}
x \oplus r_x & & y \oplus r_y & & \mathbf{s} = (x+y) \oplus (r_x + r_y) \\
\Big\Downarrow \text{Secure B-to-A} & & \Big\Downarrow & & \Big\Uparrow \text{Secure A-to-B (with LUTs)} \\
x - r_x & & y - r_y & \xrightarrow[\text{(classical)}]{+} & (x+y) - (r_x + r_y)
\end{array}
$$

- If we have only one addition (followed by boolean operations) can we avoid mask-switching ?

New method

- The new proposed algorithm is based on a more direct approacxh

$$
x \oplus r_x \qquad y \oplus r_y \qquad \xRightarrow{\text{Secure adder}} \qquad \mathbf{s} = (x+y) \oplus (r_x \oplus r_y)
$$

Outline

1 **Preliminary Background**
- DPA attacks and countermeasures
- Masking and switching method

2 **A new DPA resistant addition algorithm**
- Basic algorithm
- DPA resistant addition algorithm

3 **Application to XTEA**
- XTEA overview
- Preventing first-order DPA
- Performance analysis

4 **Conclusion**

Our construction

- The goal is to securely compute $\mathbf{S} = (x + y) \oplus r_s$ from $(\mathbf{X}, r_x)$ and $(\mathbf{Y}, r_y)$ and without compromising the x or y through DPA

Idea: $x + y = x \oplus y \oplus carry(x, y)$

- Construct an addition algorithm that takes blinded operands as input

$$\begin{aligned} \mathbf{S} = (x + y) \oplus r_s &= (x \oplus y \oplus c) \oplus r_s \\ &= (\mathbf{X} \oplus r_x) \oplus (\mathbf{Y} \oplus r_y) \oplus c \oplus r_s \\ &= \mathbf{X} \oplus \mathbf{Y} \oplus c \quad \text{by setting } r_s = r_x \oplus r_y \end{aligned}$$

- Find an algorithm that computes the carry of two variables
- Ensure that all intermediate variables do not leak information

Our construction

- The goal is to securely compute $\mathbf{S} = (x + y) \oplus r_s$ from $(\mathbf{X}, r_x)$ and $(\mathbf{Y}, r_y)$ and without compromising the x or y through DPA

Idea: $x + y = x \oplus y \oplus carry(x, y)$

- Construct an addition algorithm that takes blinded operands as input

$$\begin{aligned} \mathbf{S} = (x+y) \oplus r_s = (x \oplus y \oplus c) \oplus r_s \\ = (\mathbf{X} \oplus r_x) \oplus (\mathbf{Y} \oplus r_y) \oplus c \oplus r_s \\ = \mathbf{X} \oplus \mathbf{Y} \oplus c \quad \text{by setting } r_s = r_x \oplus r_y \end{aligned}$$

- Find an algorithm that computes the carry of two variables
- Ensure that all intermediate variables do not leak information

Our construction

- The goal is to securely compute $\mathbf{S} = (x + y) \oplus r_s$ from $(\mathbf{X}, r_x)$ and $(\mathbf{Y}, r_y)$ and without compromising the x or y through DPA

Idea: $x + y = x \oplus y \oplus carry(x, y)$

- Construct an addition algorithm that takes blinded operands as input

$$\begin{aligned} \mathbf{S} &= (x + y) \oplus r_s = (x \oplus y \oplus c) \oplus r_s \\ &= (\mathbf{X} \oplus r_x) \oplus (\mathbf{Y} \oplus r_y) \oplus c \oplus r_s \\ &= \mathbf{X} \oplus \mathbf{Y} \oplus c \quad \text{by setting } r_s = r_x \oplus r_y \end{aligned}$$

- Find an algorithm that computes the carry of two variables
- Ensure that all intermediate variables do not leak information

technicolor

Addition algorithm

- AND-XOR-and-double method

Input: $(x, y) \in \mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$
Output: $s = x + y \pmod{2^k}$

1. $\mathrm{A} \leftarrow x \oplus y$; $\mathrm{B} \leftarrow x \,\&\, y$; $\mathrm{C} \leftarrow 0$
2. For $i = 1$ to $k - 1$ do
   - $\mathrm{C} \leftarrow \mathrm{C} \,\&\, \mathrm{A}$
   - $\mathrm{C} \leftarrow \mathrm{C} \oplus \mathrm{B}$
   - $\mathrm{C} \leftarrow 2{\cdot}\mathrm{C}$
3. $\mathrm{A} \leftarrow \mathrm{A} \oplus \mathrm{C}$
4. Return A

- Right-to-left carry evaluation
- The carry is iteratively computed using A, B
- Basis of our construction

Addition algorithm

- AND-XOR-and-double method

Input: $(x, y) \in \mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$
Output: $s = x + y \pmod{2^k}$

1. $A \leftarrow x \oplus y$; $B \leftarrow x \,\&\, y$; $C \leftarrow 0$
2. For $i = 1$ to $k - 1$ do
   - $C \leftarrow C \,\&\, A$
   - $C \leftarrow C \oplus B$
   - $C \leftarrow 2{\cdot}C$
3. $A \leftarrow A \oplus C$
4. Return A

- Right-to-left carry evaluation
- The carry is iteratively computed using A, B
- Basis of our construction

Addition algorithm

- AND-XOR-and-double method

Input: $(x, y) \in \mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$
Output: $s = x + y \pmod{2^k}$

1. $\mathrm{A} \leftarrow x \oplus y$; $\mathrm{B} \leftarrow x \mathbin{\&} y$; $\mathrm{C} \leftarrow 0$
2. For $i = 1$ to $k - 1$ do
   - $\mathrm{C} \leftarrow \mathrm{C} \mathbin{\&} \mathrm{A}$
   - $\mathrm{C} \leftarrow \mathrm{C} \oplus \mathrm{B}$
   - $\mathrm{C} \leftarrow 2 \cdot \mathrm{C}$
3. $\mathrm{A} \leftarrow \mathrm{A} \oplus \mathrm{C}$
4. Return A

- Right-to-left carry evaluation
- The carry is iteratively computed using A, B
- Basis of our construction

Addition algorithm

- AND-XOR-and-double method

Input: $(x, y) \in \mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$
Output: $s = x + y \pmod{2^k}$

1. $\mathrm{A} \leftarrow x \oplus y$; $\mathrm{B} \leftarrow x \,\&\, y$; $\mathrm{C} \leftarrow 0$
2. For $i = 1$ to $k - 1$ do
   - $\mathrm{C} \leftarrow \mathrm{C} \,\&\, \mathrm{A}$
   - $\mathrm{C} \leftarrow \mathrm{C} \oplus \mathrm{B}$
   - $\mathrm{C} \leftarrow 2 \cdot \mathrm{C}$
3. $\mathrm{A} \leftarrow \mathrm{A} \oplus \mathrm{C}$
4. Return A

- Right-to-left carry evaluation
- The carry is iteratively computed using A, B
- Basis of our construction

Secure addition

Addition with blinded operands

Input: $(\mathbf{X} = x \oplus r_x, \mathbf{Y} = y \oplus r_y, r_x, r_y, \gamma) \in \mathbb{Z}_{2^k}^5$
Output: $(\mathbf{S} = (x + y) \oplus r_s, r_s = r_x \oplus r_y)$

▷ Initialization
$\mathrm{B}_0 \leftarrow \gamma \oplus \mathbf{X} \,\&\, \mathbf{Y}$; $\mathrm{T} \leftarrow \mathbf{X} \,\&\, r_y$;
$\mathrm{B}_0 \leftarrow \mathrm{B}_0 \oplus \mathrm{T}$; $\mathrm{T} \leftarrow \mathbf{Y} \,\&\, r_x$;
$\mathrm{B}_0 \leftarrow \mathrm{B}_0 \oplus \mathrm{T}$; $\mathrm{T} \leftarrow r_x \,\&\, r_y$;
$\mathrm{B}_0 \leftarrow \mathrm{B}_0 \oplus \mathrm{T}$ ▷ $\mathrm{B}_0 = x \,\&\, y \oplus \gamma$
$\mathrm{A}_0 \leftarrow \mathbf{X} \oplus \mathbf{Y}$; $\mathrm{A}_1 \leftarrow r_x \oplus r_y$;
$\mathrm{C}_0 \leftarrow 2 \cdot \gamma$; $\mathrm{C}_1 \leftarrow 2 \cdot \gamma$;
$\Omega \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_0 \oplus \mathrm{B}_0$;
$\Omega \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_1 \oplus \Omega$;
$\mathrm{C}_0 \leftarrow 2 \cdot \mathrm{B}_0$;

▷ Main loop
for **to** $k - 1$ **do**
  $\mathrm{C}_0 \leftarrow \mathrm{C}_0 \oplus \Omega$;
  $\mathrm{C}_0 \leftarrow 2 \cdot \mathrm{C}_0$ ▷ $\mathrm{C}_0 = C \oplus 2\gamma$
end

▷ Aggregation
$\mathrm{A}_0 \leftarrow \mathrm{A}_0 \oplus \mathrm{C}_0$; $\mathrm{A}_0 \leftarrow \mathrm{A}_0 \oplus \mathrm{C}_1$ ▷ $\mathrm{A}_0 = \mathbf{X} \oplus \mathbf{Y} \oplus C$

return $(\mathrm{A}_0, \mathrm{A}_1)$

Basic addition

Input: $(x, y) \in \mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$
Output: $s = x + y \pmod{2^k} = x \oplus y \oplus \text{carry}$

▷ Initialization
$\mathrm{B} \leftarrow x \,\&\, y$;
$\mathrm{A} \leftarrow x \oplus y$;
$\mathrm{C} \leftarrow 0$;

▷ Main loop
for $i = 1$ **to** $k - 1$ **do**
  $\mathrm{C} \leftarrow \mathrm{C} \,\&\, \mathrm{A}$;
  $\mathrm{C} \leftarrow \mathrm{C} \oplus \mathrm{B}$;
  $\mathrm{C} \leftarrow 2 \cdot \mathrm{C}$;
end

▷ Aggregation
$\mathrm{A} \leftarrow \mathrm{A} \oplus \mathrm{C}$;

return A

technicolor

Secure addition

Addition with blinded operands

Input: $(\mathbf{X} = x \oplus r_x, \mathbf{Y} = y \oplus r_y, r_x, r_y, \gamma) \in \mathbb{Z}_{2^k}^5$
Output: $(\mathbf{S} = (x + y) \oplus r_s, r_s = r_x \oplus r_y)$

▷ `Initialization`
$B_0 \leftarrow \gamma \oplus \mathbf{X} \,\&\, \mathbf{Y}$; $T \leftarrow \mathbf{X} \,\&\, r_y$;
$B_0 \leftarrow B_0 \oplus T$; $T \leftarrow \mathbf{Y} \,\&\, r_x$;
$B_0 \leftarrow B_0 \oplus T$; $T \leftarrow r_x \,\&\, r_y$;
$B_0 \leftarrow B_0 \oplus T$ ▷ $B_0 = x \,\&\, y \oplus \gamma$
$A_0 \leftarrow \mathbf{X} \oplus \mathbf{Y}$; $A_1 \leftarrow r_x \oplus r_y$;
$C_0 \leftarrow 2 \cdot \gamma$; $C_1 \leftarrow 2 \cdot \gamma$;
$\Omega \leftarrow C_0 \,\&\, A_0 \oplus B_0$;
$\Omega \leftarrow C_0 \,\&\, A_1 \oplus \Omega$;
$C_0 \leftarrow 2 \cdot B_0$;

▷ `Main loop`
for to $k - 1$ **do**
  $C_0 \leftarrow C_0 \oplus \Omega$;
  $C_0 \leftarrow 2 \cdot C_0$ ▷ $C_0 = C \oplus 2\gamma$
end

▷ `Aggregation`
$A_0 \leftarrow A_0 \oplus C_0$; $A_0 \leftarrow A_0 \oplus C_1$ ▷ $A_0 = \mathbf{X} \oplus \mathbf{Y} \oplus C$
return (A_0, A_1)

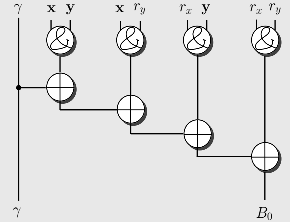
Basic addition

Input: $(x, y) \in \mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$
Output: $s = x + y \pmod{2^k} = x \oplus y \oplus \text{carry}$

▷ `Initialization`
$B \leftarrow x \,\&\, y$;

Trichina trick for secure AND

- Series of 4 AND and 4 XOR



technicolor

Secure addition

Addition with blinded operands

Input: $(\mathbf{X} = x \oplus r_x, \mathbf{Y} = y \oplus r_y, r_x, r_y, \gamma) \in \mathbb{Z}_{2^k}^5$
Output: $(\mathbf{S} = (x + y) \oplus r_s, r_s = r_x \oplus r_y)$

▷ `Initialization`
$B_0 \leftarrow \gamma \oplus \mathbf{X} \,\&\, \mathbf{Y}$; $T \leftarrow \mathbf{X} \,\&\, r_y$;
$B_0 \leftarrow B_0 \oplus T$; $T \leftarrow \mathbf{Y} \,\&\, r_x$;
$B_0 \leftarrow B_0 \oplus T$; $T \leftarrow r_x \,\&\, r_y$;
$B_0 \leftarrow B_0 \oplus T$ ▷ $B_0 = x \,\&\, y \oplus \gamma$
$A_0 \leftarrow \mathbf{X} \oplus \mathbf{Y}$; $A_1 \leftarrow r_x \oplus r_y$;
$C_0 \leftarrow 2 \cdot \gamma$; $C_1 \leftarrow 2 \cdot \gamma$;
$\Omega \leftarrow C_0 \,\&\, A_0 \oplus B_0$;
$\Omega \leftarrow C_0 \,\&\, A_1 \oplus \Omega$;
$C_0 \leftarrow 2 \cdot B_0$;

▷ `Main loop`
for to $k - 1$ **do**
  $C_0 \leftarrow C_0 \oplus \Omega$;
  $C_0 \leftarrow 2 \cdot C_0$ ▷ $C_0 = C \oplus 2\gamma$
end

▷ `Aggregation`
$A_0 \leftarrow A_0 \oplus C_0$; $A_0 \leftarrow A_0 \oplus C_1$ ▷ $A_0 = \mathbf{X} \oplus \mathbf{Y} \oplus C$

return (A_0, A_1)

Basic addition

Input: $(x, y) \in \mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$
Output: $s = x + y \pmod{2^k} = x \oplus y \oplus \text{carry}$

▷ `Initialization`
$B \leftarrow x \,\&\, y$;
$A \leftarrow x \oplus y$;
$C \leftarrow 0$;

▷ `Main loop`
for $i = 1$ **to** $k - 1$ **do**
  $C \leftarrow C \,\&\, A$;
  $C \leftarrow C \oplus B$;
  $C \leftarrow 2 \cdot C$;
end

▷ `Aggregation`
$A \leftarrow A \oplus C$;

return A

technicolor

Secure addition

Addition with blinded operands

Input: $(\mathbf{X} = x \oplus r_x, \mathbf{Y} = y \oplus r_y, r_x, r_y, \gamma) \in \mathbb{Z}_{2^k}^5$
Output: $(\mathbf{S} = (x + y) \oplus r_s, r_s = r_x \oplus r_y)$

▷ `Initialization`

$B_0 \leftarrow \gamma \oplus \mathbf{X} \,\&\, \mathbf{Y}$; $T \leftarrow \mathbf{X} \,\&\, r_y$;
$B_0 \leftarrow B_0 \oplus T$; $T \leftarrow \mathbf{Y} \,\&\, r_x$;
$B_0 \leftarrow B_0 \oplus T$; $T \leftarrow r_x \,\&\, r_y$;
$B_0 \leftarrow B_0 \oplus T$ ▷ $B_0 = x \,\&\, y \oplus \gamma$
$A_0 \leftarrow \mathbf{X} \oplus \mathbf{Y}$; $A_1 \leftarrow r_x \oplus r_y$;
$C_0 \leftarrow 2 \cdot \gamma$; $C_1 \leftarrow 2 \cdot \gamma$;
$\Omega \leftarrow C_0 \,\&\, A_0 \oplus B_0$;
$\Omega \leftarrow C_0 \,\&\, A_1 \oplus \Omega$;
$C_0 \leftarrow 2 \cdot B_0$;

▷ `Main loop`

for $i = 1$ **to** $k - 1$ **do**
  $C_0 \leftarrow C_0 \,\&\, A$;
  $C_0 \leftarrow C_0 \oplus \Omega$;
  $C_0 \leftarrow 2 \cdot C_0$ ▷ $C_0 = C \oplus 2\gamma$
end

▷ `Aggregation`

$A_0 \leftarrow A_0 \oplus C_0$; $A_0 \leftarrow A_0 \oplus C_1$ ▷ $A_0 = \mathbf{X} \oplus \mathbf{Y} \oplus C$
return (A_0, A_1)

Basic addition

Input: $(x, y) \in \mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$
Output: $s = x + y \pmod{2^k} = x \oplus y \oplus \text{carry}$

▷ `Initialization`

$B \leftarrow x \,\&\, y$;
$A \leftarrow x \oplus y$;
$C \leftarrow 0$;

Goubin's trick for carry masking

- Mask the carry C with 2γ
- Pre-compute the loop transformed mask Ω

$$\begin{aligned} \Omega &= 2\gamma \,\&\, A \oplus B \oplus \gamma \\ &= 2\gamma \,\&\, A_0 \oplus B_0 \oplus 2\gamma \,\&\, A_1 \end{aligned}$$

Secure addition

Addition with blinded operands

Input: $(\mathbf{X} = x \oplus r_x, \mathbf{Y} = y \oplus r_y, r_x, r_y, \gamma) \in \mathbb{Z}_{2^k}^5$
Output: $(\mathbf{S} = (x + y) \oplus r_s, r_s = r_x \oplus r_y)$

$\triangleright$ `Initialization`
$B_0 \leftarrow \gamma \oplus \mathbf{X} \,\&\, \mathbf{Y}$; $T \leftarrow \mathbf{X} \,\&\, r_y$;
$B_0 \leftarrow B_0 \oplus T$; $T \leftarrow \mathbf{Y} \,\&\, r_x$;
$B_0 \leftarrow B_0 \oplus T$; $T \leftarrow r_x \,\&\, r_y$;
$B_0 \leftarrow B_0 \oplus T$ $\qquad \triangleright B_0 = x \,\&\, y \oplus \gamma$
$A_0 \leftarrow \mathbf{X} \oplus \mathbf{Y}$; $A_1 \leftarrow r_x \oplus r_y$;
$C_0 \leftarrow 2 \cdot \gamma$; $C_1 \leftarrow 2 \cdot \gamma$;
$\Omega \leftarrow C_0 \,\&\, A_0 \oplus B_0$;
$\Omega \leftarrow C_0 \,\&\, A_1 \oplus \Omega$;
$C_0 \leftarrow 2 \cdot B_0$;

$\triangleright$ `Main loop`
for $i = 1$ **to** $k - 1$ **do**
$\quad C_0 \leftarrow C_0 \,\&\, A$;
$\quad C_0 \leftarrow C_0 \oplus \Omega$;
$\quad C_0 \leftarrow 2 \cdot C_0$ $\qquad \triangleright C_0 = C \oplus 2\gamma$
end

$\triangleright$ `Aggregation`
$A_0 \leftarrow A_0 \oplus C_0$; $A_0 \leftarrow A_0 \oplus C_1$ $\qquad \triangleright A_0 = \mathbf{X} \oplus \mathbf{Y} \oplus C$
return (A_0, A_1)

Basic addition

Input: $(x, y) \in \mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$
Output: $s = x + y \pmod{2^k} = x \oplus y \oplus \text{carry}$

$\triangleright$ `Initialization`
$B \leftarrow x \,\&\, y$;
$A \leftarrow x \oplus y$;
$C \leftarrow 0$;

$\triangleright$ `Main loop`
for $i = 1$ **to** $k - 1$ **do**
$\quad C \leftarrow C \,\&\, A$;
$\quad C \leftarrow C \oplus B$;
$\quad C \leftarrow 2 \cdot C$;
end

$\triangleright$ `Aggregation`
$A \leftarrow A \oplus C$;

return A

Secure addition

Addition with blinded operands

Input: $(\mathbf{X} = x \oplus r_x, \mathbf{Y} = y \oplus r_y, r_x, r_y, \gamma) \in \mathbb{Z}_{2^k}^5$
Output: $(\mathbf{S} = (x + y) \oplus r_s, r_s = r_x \oplus r_y)$

▷ Initialization
$B_0 \leftarrow \gamma \oplus \mathbf{X} \,\&\, \mathbf{Y}$; $T \leftarrow \mathbf{X} \,\&\, r_y$;
$B_0 \leftarrow B_0 \oplus T$; $T \leftarrow \mathbf{Y} \,\&\, r_x$;
$B_0 \leftarrow B_0 \oplus T$; $T \leftarrow r_x \,\&\, r_y$;
$B_0 \leftarrow B_0 \oplus T$ ▷ $B_0 = x \,\&\, y \oplus \gamma$
$A_0 \leftarrow \mathbf{X} \oplus \mathbf{Y}$; $A_1 \leftarrow r_x \oplus r_y$;
$C_0 \leftarrow 2 \cdot \gamma$; $C_1 \leftarrow 2 \cdot \gamma$;
$\Omega \leftarrow C_0 \,\&\, A_0 \oplus B_0$;
$\Omega \leftarrow C_0 \,\&\, A_1 \oplus \Omega$;
$C_0 \leftarrow 2 \cdot B_0$;

▷ Main loop
for $i = 1$ **to** $k - 1$ **do**
  $T \leftarrow C_0 \,\&\, A_0$;
  $C_0 \leftarrow C_0 \,\&\, A_1$;
  $C_0 \leftarrow C_0 \oplus \Omega$;
  $C_0 \leftarrow C_0 \oplus T$;
  $C_0 \leftarrow 2 \cdot C_0$ ▷ $C_0 = C \oplus 2\gamma$
end

▷ Aggregation
$A_0 \leftarrow A_0 \oplus C_0$; $A_0 \leftarrow A_0 \oplus C_1$ ▷ $A_0 = \mathbf{X} \oplus \mathbf{Y} \oplus C$

return (A_0, A_1)

Basic addition

Input: $(x, y) \in \mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$
Output: $s = x + y \pmod{2^k} = x \oplus y \oplus \text{carry}$

▷ Initialization
$B \leftarrow x \,\&\, y$;
$A \leftarrow x \oplus y$;
$C \leftarrow 0$;

▷ Main loop
for $i = 1$ **to** $k - 1$ **do**
  $C \leftarrow C \,\&\, A$;
  $C \leftarrow C \oplus B$;
  $C \leftarrow 2 \cdot C$;
end

▷ Aggregation
$A \leftarrow A \oplus C$;

return A

technicolor

Secure addition

Addition with blinded operands

Input: $(\mathbf{X} = x \oplus r_x, \mathbf{Y} = y \oplus r_y, r_x, r_y, \gamma) \in \mathbb{Z}_{2^k}^5$
Output: $(\mathbf{S} = (x + y) \oplus r_s, r_s = r_x \oplus r_y)$

▷ Initialization
$\mathrm{B}_0 \leftarrow \gamma \oplus \mathbf{X} \,\&\, \mathbf{Y}$; $\mathrm{T} \leftarrow \mathbf{X} \,\&\, r_y$;
$\mathrm{B}_0 \leftarrow \mathrm{B}_0 \oplus \mathrm{T}$; $\mathrm{T} \leftarrow \mathbf{Y} \,\&\, r_x$;
$\mathrm{B}_0 \leftarrow \mathrm{B}_0 \oplus \mathrm{T}$; $\mathrm{T} \leftarrow r_x \,\&\, r_y$;
$\mathrm{B}_0 \leftarrow \mathrm{B}_0 \oplus \mathrm{T}$ ▷ $\mathrm{B}_0 = x \,\&\, y \oplus \gamma$
$\mathrm{A}_0 \leftarrow \mathbf{X} \oplus \mathbf{Y}$; $\mathrm{A}_1 \leftarrow r_x \oplus r_y$;
$\mathrm{C}_0 \leftarrow 2 \cdot \gamma$; $\mathrm{C}_1 \leftarrow 2 \cdot \gamma$;
$\Omega \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_0 \oplus \mathrm{B}_0$;
$\Omega \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_1 \oplus \Omega$;
$\mathrm{C}_0 \leftarrow 2 \cdot \mathrm{B}_0$;

▷ Main loop
for $i = 2$ **to** $k - 1$ **do**
    $\mathrm{T} \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_0$;
    $\mathrm{C}_0 \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_1$;
    $\mathrm{C}_0 \leftarrow \mathrm{C}_0 \oplus \Omega$;
    $\mathrm{C}_0 \leftarrow \mathrm{C}_0 \oplus \mathrm{T}$;
    $\mathrm{C}_0 \leftarrow 2 \cdot \mathrm{C}_0$ ▷ $\mathrm{C}_0 = C \oplus 2\gamma$
end

▷ Aggregation
$\mathrm{A}_0 \leftarrow \mathrm{A}_0 \oplus \mathrm{C}_0$; $\mathrm{A}_0 \leftarrow \mathrm{A}_0 \oplus \mathrm{C}_1$ ▷ $\mathrm{A}_0 = \mathbf{X} \oplus \mathbf{Y} \oplus C$
return $(\mathrm{A}_0, \mathrm{A}_1)$

Basic addition

Input: $(x, y) \in \mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$
Output: $s = x + y \pmod{2^k} = x \oplus y \oplus \text{carry}$

▷ Initialization
$\mathrm{B} \leftarrow x \,\&\, y$;

A new trick

- We noted that the carry after round 1
  $C = 2 \cdot (x \,\&\, y \oplus \gamma) = 2 \cdot \mathrm{B}_0$
- We saved operations of round 1
- The trick applies also to Goubin A-to-B conversion (cost is reduced from $5k + 5$ down to $\underline{5k + 1}$ operations)

technicolor

Secure addition

Addition with blinded operands

Input: $(\mathbf{X} = x \oplus r_x, \mathbf{Y} = y \oplus r_y, r_x, r_y, \gamma) \in \mathbb{Z}_{2^k}^5$
Output: $(\mathbf{S} = (x + y) \oplus r_s, r_s = r_x \oplus r_y)$

▷ `Initialization`
$\mathrm{B}_0 \leftarrow \gamma \oplus \mathbf{X} \,\&\, \mathbf{Y}$; $\mathrm{T} \leftarrow \mathbf{X} \,\&\, r_y$;
$\mathrm{B}_0 \leftarrow \mathrm{B}_0 \oplus \mathrm{T}$; $\mathrm{T} \leftarrow \mathbf{Y} \,\&\, r_x$;
$\mathrm{B}_0 \leftarrow \mathrm{B}_0 \oplus \mathrm{T}$; $\mathrm{T} \leftarrow r_x \,\&\, r_y$;
$\mathrm{B}_0 \leftarrow \mathrm{B}_0 \oplus \mathrm{T}$ ▷ $\mathrm{B}_0 = x \,\&\, y \oplus \gamma$
$\mathrm{A}_0 \leftarrow \mathbf{X} \oplus \mathbf{Y}$; $\mathrm{A}_1 \leftarrow r_x \oplus r_y$;
$\mathrm{C}_0 \leftarrow 2 \cdot \gamma$; $\mathrm{C}_1 \leftarrow 2 \cdot \gamma$;
$\Omega \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_0 \oplus \mathrm{B}_0$;
$\Omega \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_1 \oplus \Omega$;
$\mathrm{C}_0 \leftarrow 2 \cdot \mathrm{B}_0$;

▷ `Main loop`
for $i = 2$ **to** $k - 1$ **do**
  $\mathrm{T} \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_0$;
  $\mathrm{C}_0 \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_1$;
  $\mathrm{C}_0 \leftarrow \mathrm{C}_0 \oplus \Omega$;
  $\mathrm{C}_0 \leftarrow \mathrm{C}_0 \oplus \mathrm{T}$;
  $\mathrm{C}_0 \leftarrow 2 \cdot \mathrm{C}_0$ ▷ $\mathrm{C}_0 = C \oplus 2\gamma$
end

▷ `Aggregation`
$\mathrm{A}_0 \leftarrow \mathrm{A}_0 \oplus \mathrm{C}_0$; $\mathrm{A}_0 \leftarrow \mathrm{A}_0 \oplus \mathrm{C}_1$ ▷ $\mathrm{A}_0 = \mathbf{X} \oplus \mathbf{Y} \oplus C$
return $(\mathrm{A}_0, \mathrm{A}_1)$

Basic addition

Input: $(x, y) \in \mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$
Output: $s = x + y \pmod{2^k} = x \oplus y \oplus \mathrm{carry}$

▷ `Initialization`
$\mathrm{B} \leftarrow x \,\&\, y$;
$\mathrm{A} \leftarrow x \oplus y$;
$\mathrm{C} \leftarrow 0$;

▷ `Main loop`
for $i = 1$ **to** $k - 1$ **do**
  $\mathrm{C} \leftarrow \mathrm{C} \,\&\, \mathrm{A}$;
  $\mathrm{C} \leftarrow \mathrm{C} \oplus \mathrm{B}$;
  $\mathrm{C} \leftarrow 2 \cdot \mathrm{C}$;
end

▷ `Aggregation`
$\mathrm{A} \leftarrow \mathrm{A} \oplus \mathrm{C}$;
return A

technicolor

Secure addition

Addition with blinded operands

Input: $(\mathbf{X} = x \oplus r_x, \mathbf{Y} = y \oplus r_y, r_x, r_y, \gamma) \in \mathbb{Z}_{2^k}^5$
Output: $(\mathbf{S} = (x + y) \oplus r_s, r_s = r_x \oplus r_y)$

$\triangleright$ `Initialization`
$\mathrm{B}_0 \leftarrow \gamma \oplus \mathbf{X} \,\&\, \mathbf{Y}$; $\mathrm{T} \leftarrow \mathbf{X} \,\&\, r_y$;
$\mathrm{B}_0 \leftarrow \mathrm{B}_0 \oplus \mathrm{T}$; $\mathrm{T} \leftarrow \mathbf{Y} \,\&\, r_x$;
$\mathrm{B}_0 \leftarrow \mathrm{B}_0 \oplus \mathrm{T}$; $\mathrm{T} \leftarrow r_x \,\&\, r_y$;
$\mathrm{B}_0 \leftarrow \mathrm{B}_0 \oplus \mathrm{T}$ $\triangleright\, \mathrm{B}_0 = x \,\&\, y \oplus \gamma$
$\mathrm{A}_0 \leftarrow \mathbf{X} \oplus \mathbf{Y}$; $\mathrm{A}_1 \leftarrow r_x \oplus r_y$;
$\mathrm{C}_0 \leftarrow 2 \cdot \gamma$; $\mathrm{C}_1 \leftarrow 2 \cdot \gamma$;
$\Omega \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_0 \oplus \mathrm{B}_0$;
$\Omega \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_1 \oplus \Omega$;
$\mathrm{C}_0 \leftarrow 2 \cdot \mathrm{B}_0$;

$\triangleright$ `Main loop`
for $i = 2$ **to** $k - 1$ **do**
- $\mathrm{T} \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_0$;
- $\mathrm{C}_0 \leftarrow \mathrm{C}_0 \,\&\, \mathrm{A}_1$;
- $\mathrm{C}_0 \leftarrow \mathrm{C}_0 \oplus \Omega$;
- $\mathrm{C}_0 \leftarrow \mathrm{C}_0 \oplus \mathrm{T}$;
- $\mathrm{C}_0 \leftarrow 2 \cdot \mathrm{C}_0$ $\triangleright\, \mathrm{C}_0 = C \oplus 2\gamma$

end

$\triangleright$ `Aggregation`
$\mathrm{A}_0 \leftarrow \mathrm{A}_0 \oplus \mathrm{C}_0$; $\mathrm{A}_0 \leftarrow \mathrm{A}_0 \oplus \mathrm{C}_1$ $\triangleright\, \mathrm{A}_0 = \mathbf{X} \oplus \mathbf{Y} \oplus C$
return $(\mathrm{A}_0, \mathrm{A}_1)$

Basic addition

Input: $(x, y) \in \mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$
Output: $s = x + y \pmod{2^k} = x \oplus y \oplus \text{carry}$

$\triangleright$ `Initialization`
$\mathrm{B} \leftarrow x \,\&\, y$;

Final algorithm

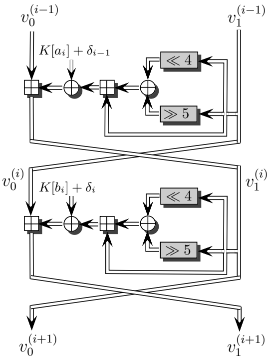
- We rearranged the operations to obtain a better memory management
- We also save a few more operations
- The final cost $\underline{5k + 8}$ basic ops
- Faster than Goubin's method ($\underline{5k + 21}$ ops)

Outline

technicolor

XTEA overview

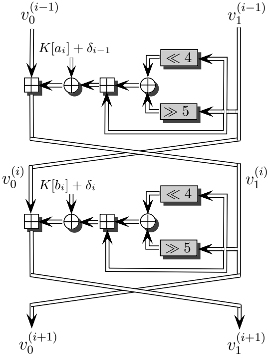
- XTEA is a lightweight cipher designed by Needham and Wheeler
- 32 rounds, 128-bit key length, 64-bit block length
- Minimal key set-up: 32-bit part of the key used in each round
- Security: combination of additions, shifts and XORs
- Simple routine: Feistel structure with 32-bit word inputs (v_0, v_1), without S-box



technicolor

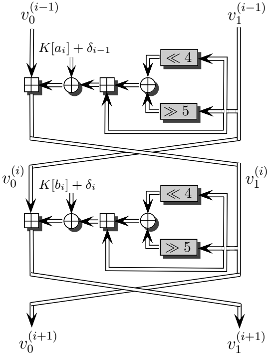
XTEA overview

- XTEA is a lightweight cipher designed by Needham and Wheeler
- 32 rounds, 128-bit key length, 64-bit block length
- Minimal key set-up: 32-bit part of the key used in each round
- Security: combination of additions, shifts and XORs
- Simple routine: Feistel structure with 32-bit word inputs (v_0, v_1), without S-box



XTEA overview

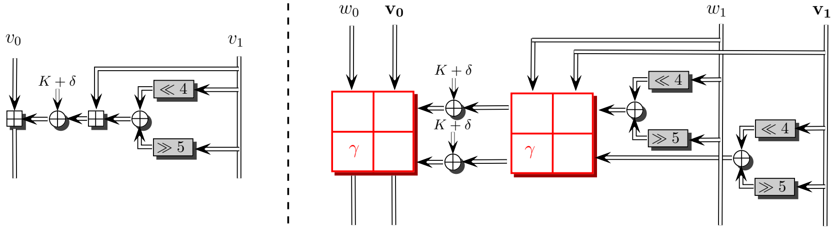
- XTEA is a lightweight cipher designed by Needham and Wheeler
- 32 rounds, 128-bit key length, 64-bit block length
- Minimal key set-up: 32-bit part of the key used in each round
- Security: combination of additions, shifts and XORs
- Simple routine: Feistel structure with 32-bit word inputs (v_0, v_1), without S-box



technicolor

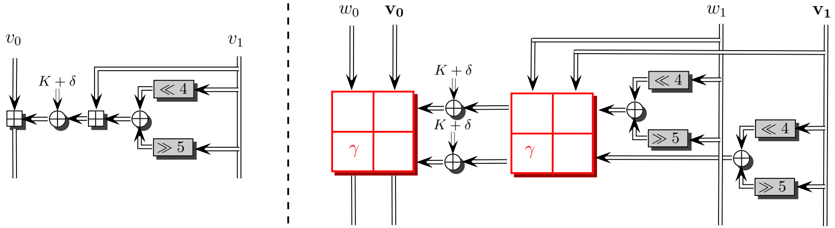
Preventing first-order DPA

- Fresh 32-bit random masks w_0 , w_1 and γ are used for each encryption process
  - $\mathbf{V_0} = v_0 \oplus w_0$, $\mathbf{V_1} = v_1 \oplus w_1$, γ is used with the secure addition algorithm
  - Operations on the masked variables and the masks are processed separately
  - The same masks are maintained across all rounds
  - At the end the masks (w_0 , w_1) enable to get the unmasked ciphertext



Preventing first-order DPA

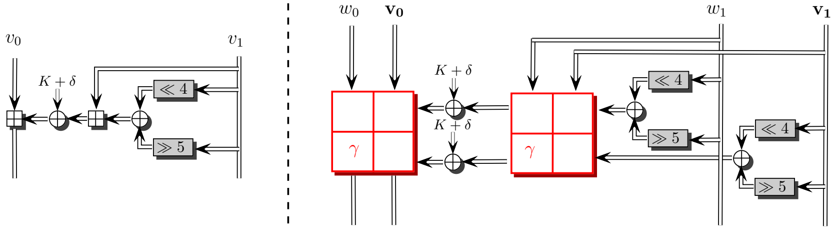
- Fresh 32-bit random masks w_0 , w_1 and γ are used for each encryption process
  - $\mathbf{V_0} = v_0 \oplus w_0$,$\mathbf{V_1} = v_1 \oplus w_1$, γ is used with the secure addition algorithm
  - Operations on the masked variables and the masks are processed separately
  - The same masks are maintained across all rounds
  - At the end the masks (w_0 ,w_1) enable to get the unmasked ciphertext



technicolor

Preventing first-order DPA

- Fresh 32-bit random masks w_0 , w_1 and γ are used for each encryption process
  - $\mathbf{V_0} = v_0 \oplus w_0$,$\mathbf{V_1} = v_1 \oplus w_1$, γ is used with the secure addition algorithm
  - Operations on the masked variables and the masks are processed separately
  - The same masks are maintained across all rounds
  - At the end the masks (w_0 ,w_1) enable to get the unmasked ciphertext



technicolor

Performance Analysis

Algorithms	ROM [bytes]	RAM [bytes]	Cycles/byte
XTEA	114	16	60
masked XTEA (New alg.)	**379**	**28**	**2410**
" (Optimized Goubin)	395 (+4%)	**28**	**2515 (+4%)**
" (Neiβe and Pulkus '04)	620 (+39%)	45	3180 (+24%)
" (Debraize '12)	664 (+43%)	51	3403 (+29%)

- Goal: implementation of protected XTEA using different algorithms with the smallest memory footprint
  - The nibble size tested is $k = 4$ with LUTs methods
  - An optimized version of the Goubin method was implemented for the tests (see Appendix in the paper)
  - C code, a 32-bit Intel based processor used for evaluation
  - The compilation options were chosen to favor small code size
- New method is compact and fast
  - Saves at least 39% of the memory space compared to methods based on LUTs
  - Up to 29% faster than LUTs methods

Performance Analysis

Algorithms	ROM [bytes]	RAM [bytes]	Cycles/byte
XTEA	114	16	60
masked XTEA (New alg.)	**379**	**28**	**2410**
" (Optimized Goubin)	395 (+4%)	**28**	**2515 (+4%)**
" (Neiβe and Pulkus '04)	620 (+39%)	45	3180 (+24%)
" (Debraize '12)	664 (+43%)	51	3403 (+29%)

- Goal: implementation of protected XTEA using different algorithms with the smallest memory footprint
  - The nibble size tested is $k = 4$ with LUTs methods
  - An optimized version of the Goubin method was implemented for the tests (see Appendix in the paper)
  - C code, a 32-bit Intel based processor used for evaluation
  - The compilation options were chosen to favor small code size
- New method is compact and fast
  - Saves at least 39% of the memory space compared to methods based on LUTs
  - Up to 29% faster than LUTs methods

Outline

Summary

- Compact methods for adding 2 boolean masked variables
  - We devised a new addition algorithm
  - Approach differs from known switching methods
- Application of new addition algorithm
  - Is efficient when **one** addition occur with any operation that is compatible with boolean masking (boolean op., shift or rotation).
  - Applies to ARX based cryptosystems (XTEA, SKEIN, SAFER, etc)
- Security
  - Randomized, regular, transformed masking method
  - Protected against first-order DPA attacks
- Attractive for smartcards
  - Minimal memory footprint
  - XTEA's countermeasure and tests proved that it is well adapted to 32-bit cpus
  - With smaller word size (eg. 8-bit), the gain in speed is even more significant

Summary

- Compact methods for adding 2 boolean masked variables
  - We devised a new addition algorithm
  - Approach differs from known switching methods
- Application of new addition algorithm
  - Is efficient when **one** addition occur with any operation that is compatible with boolean masking (boolean op., shift or rotation).
  - Applies to ARX based cryptosystems (XTEA, SKEIN, SAFER, etc)
- Security
  - Randomized, regular, transformed masking method
  - Protected against first-order DPA attacks
- Attractive for smartcards
  - Minimal memory footprint
  - XTEA's countermeasure and tests proved that it is well adapted to 32-bit cpus
  - With smaller word size (eg. 8-bit), the gain in speed is even more significant

Summary

- Compact methods for adding 2 boolean masked variables
  - We devised a new addition algorithm
  - Approach differs from known switching methods
- Application of new addition algorithm
  - Is efficient when **one** addition occur with any operation that is compatible with boolean masking (boolean op., shift or rotation).
  - Applies to ARX based cryptosystems (XTEA, SKEIN, SAFER, etc)
- Security
  - Randomized, regular, transformed masking method
  - Protected against first-order DPA attacks
- Attractive for smartcards
  - Minimal memory footprint
  - XTEA's countermeasure and tests proved that it is well adapted to 32-bit cpus
  - With smaller word size (eg. 8-bit), the gain in speed is even more significant

technicolor