Faster Mask Conversion with Lookup Tables

Praveen Kumar Vadnala   Johann Großschädl

University of Luxembourg

COSADE, 2015. Berlin, Germany.

Masking

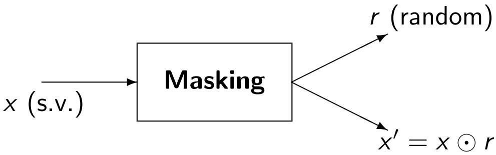
- Masking
  - Each sensitive variable is masked with a random value

  - Security can be proved
- Higher-order masking

$$
\begin{aligned}
x &\leftarrow (x_1 \odot x_2 \odot \cdots \odot x_{d+1}) \\
(x_1, \cdots, x_d) &\leftarrow \mathrm{rand}() \\
x_{d+1} &\leftarrow x \odot x_1 \odot x_2 \odot \cdots \odot x_d
\end{aligned}
$$

Masking

- Masking
  - Each sensitive variable is masked with a random value

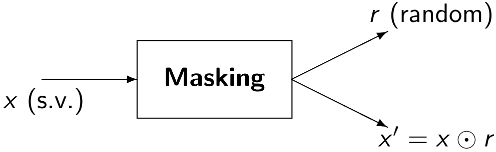


  - Security can be proved
- Higher-order masking

$$
\begin{aligned}
x &\leftarrow (x_1 \odot x_2 \odot \cdots \odot x_{d+1}) \\
(x_1, \cdots, x_d) &\leftarrow \mathrm{rand}() \\
x_{d+1} &\leftarrow x \odot x_1 \odot x_2 \odot \cdots \odot x_d
\end{aligned}
$$

Masking

- Masking
  - Each sensitive variable is masked with a random value

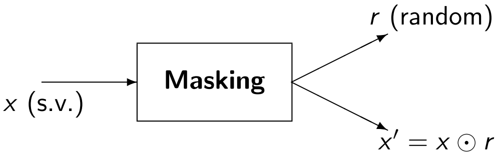


  - Security can be proved
- Higher-order masking

$$
\begin{aligned}
x &\leftarrow (x_1 \odot x_2 \odot \cdots \odot x_{d+1}) \\
(x_1, \cdots, x_d) &\leftarrow \mathrm{rand}() \\
x_{d+1} &\leftarrow x \odot x_1 \odot x_2 \odot \cdots \odot x_d
\end{aligned}
$$

Masking

- Masking
  - Each sensitive variable is masked with a random value



  - Security can be proved
- Higher-order masking

$$
\begin{aligned}
x &\leftarrow (x_1 \odot x_2 \odot \cdots \odot x_{d+1}) \\
(x_1, \cdots, x_d) &\leftarrow \mathrm{rand}() \\
x_{d+1} &\leftarrow x \odot x_1 \odot x_2 \odot \cdots \odot x_d
\end{aligned}
$$

Masking types

- Boolean masking

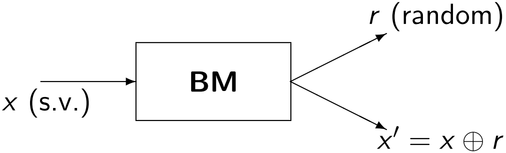
- Arithmetic masking

- Multiplicative masking

$$x : (x.r^{-1}, r)$$

Masking types

- Boolean masking



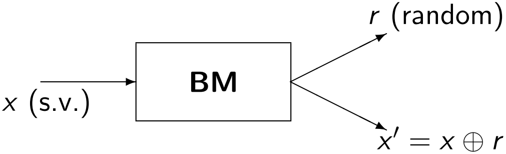
- Arithmetic masking

- Multiplicative masking

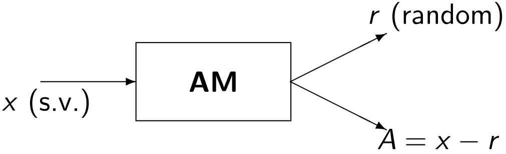
$$x : (x.r^{-1}, r)$$

Masking types

- Boolean masking



- Arithmetic masking

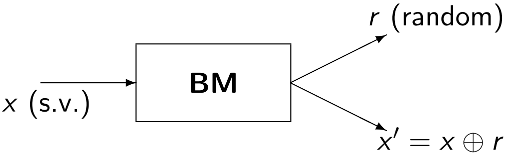


- Multiplicative masking

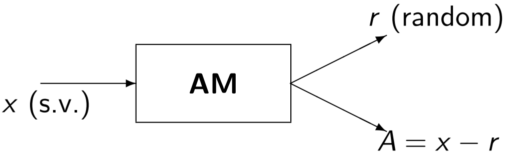
$$x : (x.r^{-1}, r)$$

Masking types

- Boolean masking



- Arithmetic masking



- Multiplicative masking

$$x : (x.r^{-1}, r)$$

Mask conversion

- Conversion problem
- This talk : Conversion between arithmetic and Boolean masking
- Applications: IDEA, HMAC-SHA1, ARX based ciphers, GOST, ...
- Two approaches to find solution
    - Convert from one form to the other
    - Perform addition directly on Boolean shares

Mask conversion

- Conversion problem
- This talk : Conversion between arithmetic and Boolean masking
- Applications: IDEA, HMAC-SHA1, ARX based ciphers, GOST, ...
- Two approaches to find solution
  - Convert from one form to the other
  - Perform addition directly on Boolean shares

Mask conversion

- Conversion problem
- This talk : Conversion between arithmetic and Boolean masking
- Applications: IDEA, HMAC-SHA1, ARX based ciphers, GOST, ...
- Two approaches to find solution
  - Convert from one form to the other
  - Perform addition directly on Boolean shares

Mask conversion

- Conversion problem
- This talk : Conversion between arithmetic and Boolean masking
- Applications: IDEA, HMAC-SHA1, ARX based ciphers, GOST, ...
- Two approaches to find solution
  - Convert from one form to the other
  - Perform addition directly on Boolean shares

Mask conversion

- Conversion problem
- This talk : Conversion between arithmetic and Boolean masking
- Applications: IDEA, HMAC-SHA1, ARX based ciphers, GOST, ...
- Two approaches to find solution
  - Convert from one form to the other
  - Perform addition directly on Boolean shares

Mask conversion

- Conversion problem
- This talk : Conversion between arithmetic and Boolean masking
- Applications: IDEA, HMAC-SHA1, ARX based ciphers, GOST, ...
- Two approaches to find solution
  - Convert from one form to the other
  - Perform addition directly on Boolean shares

State of the art

- Several solutions exist for first-order secure conversion with varying complexity
- Coron-Großschädl-Vadnala higher-order conversion
  - Based on Ishai-Sahai-Wagner method
  - Requires $2t + 1$ shares for t-th order security
- Vadnala-Großschädl second-order solution (LUT)
  - Based on generic second-order masking scheme by Prouff-Rivain
  - Needs only 3 shares for second-order security
  - Requires 2^n LUT for n-bit conversion

State of the art

- Several solutions exist for first-order secure conversion with varying complexity
- Coron-Großschädl-Vadnala higher-order conversion
  - Based on Ishai-Sahai-Wagner method
  - Requires $2t + 1$ shares for t-th order security
- Vadnala-Großschädl second-order solution (LUT)
  - Based on generic second-order masking scheme by Prouff-Rivain
  - Needs only 3 shares for second-order security
  - Requires 2^n LUT for n-bit conversion

State of the art

- Several solutions exist for first-order secure conversion with varying complexity
- Coron-Großschädl-Vadnala higher-order conversion
  - Based on Ishai-Sahai-Wagner method
  - Requires $2t + 1$ shares for t-th order security
- Vadnala-Großschädl second-order solution (LUT)
  - Based on generic second-order masking scheme by Prouff-Rivain
  - Needs only 3 shares for second-order security
  - Requires 2^n LUT for n-bit conversion

State of the art

- Several solutions exist for first-order secure conversion with varying complexity
- Coron-Großschädl-Vadnala higher-order conversion
  - Based on Ishai-Sahai-Wagner method
  - Requires $2t + 1$ shares for t-th order security
- Vadnala-Großschädl second-order solution (LUT)
  - Based on generic second-order masking scheme by Prouff-Rivain
  - Needs only 3 shares for second-order security
  - Requires 2^n LUT for n-bit conversion

State of the art

- Several solutions exist for first-order secure conversion with varying complexity
- Coron-Großschädl-Vadnala higher-order conversion
  - Based on Ishai-Sahai-Wagner method
  - Requires $2t + 1$ shares for t-th order security
- Vadnala-Großschädl second-order solution (LUT)
  - Based on generic second-order masking scheme by Prouff-Rivain
  - Needs only 3 shares for second-order security
  - Requires 2^n LUT for n-bit conversion

State of the art

- Several solutions exist for first-order secure conversion with varying complexity
- Coron-Großschädl-Vadnala higher-order conversion
  - Based on Ishai-Sahai-Wagner method
  - Requires $2t + 1$ shares for t-th order security
- Vadnala-Großschädl second-order solution (LUT)
  - Based on generic second-order masking scheme by Prouff-Rivain
  - Needs only 3 shares for second-order security
  - Requires 2^n LUT for n-bit conversion

State of the art

- Several solutions exist for first-order secure conversion with varying complexity
- Coron-Großschädl-Vadnala higher-order conversion
  - Based on Ishai-Sahai-Wagner method
  - Requires $2t + 1$ shares for t-th order security
- Vadnala-Großschädl second-order solution (LUT)
  - Based on generic second-order masking scheme by Prouff-Rivain
  - Needs only 3 shares for second-order security
  - Requires 2^n LUT for n-bit conversion

State of the art

- Several solutions exist for first-order secure conversion with varying complexity
- Coron-Großschädl-Vadnala higher-order conversion
  - Based on Ishai-Sahai-Wagner method
  - Requires $2t + 1$ shares for t-th order security
- Vadnala-Großschädl second-order solution (LUT)
  - Based on generic second-order masking scheme by Prouff-Rivain
  - Needs only 3 shares for second-order security
  - Requires 2^n LUT for n-bit conversion

Our contributions

- Improved algorithms for second-order conversion using LUT (3 shares)
- First-order secure addition (also using LUT)
- Over 85% improvement in execution time for second-order
- Application to HMAC-SHA-1 ($k = 32$)

Our contributions

- Improved algorithms for second-order conversion using LUT (3 shares)
- First-order secure addition (also using LUT)
- Over 85% improvement in execution time for second-order
- Application to HMAC-SHA-1 ($k = 32$)

Our contributions

- Improved algorithms for second-order conversion using LUT (3 shares)
- First-order secure addition (also using LUT)
- Over 85% improvement in execution time for second-order
- Application to HMAC-SHA-1 ($k = 32$)

Our contributions

- Improved algorithms for second-order conversion using LUT (3 shares)
- First-order secure addition (also using LUT)
- Over 85% improvement in execution time for second-order
- Application to HMAC-SHA-1 ($k = 32$)

Generic 2O-secure masking (Prouff-Rivain FSE, 2008)

- Input: $(x_1 = x \oplus x_1 \oplus x_2, x_2, x_3)$
- Output: $(y_1, y_2, S(x) \oplus y_1 \oplus y_2)$
- Randomizes the index $a' = a \oplus r \oplus x_2 \oplus x_3$ for $0 \leq a \leq 2^n - 1$
- Shifts the table by y_1, y_2 in one step

Generic 2O-secure masking (Prouff-Rivain FSE, 2008)

- Input: $(x_1 = x \oplus x_1 \oplus x_2, x_2, x_3)$
- Output: $(y_1, y_2, S(x) \oplus y_1 \oplus y_2)$
- Randomizes the index $a' = a \oplus r \oplus x_2 \oplus x_3$ for $0 \leq a \leq 2^n - 1$
- Shifts the table by y_1, y_2 in one step

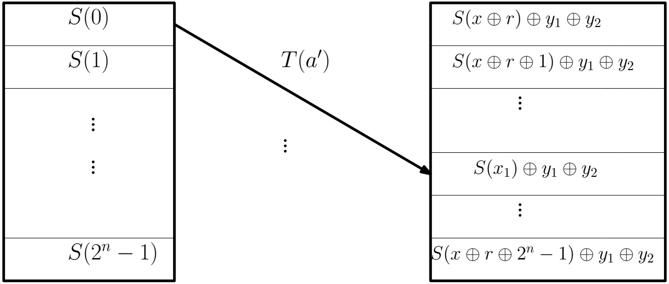
Generic 2O-secure masking (Prouff-Rivain FSE, 2008)

- Input: $(x_1 = x \oplus x_1 \oplus x_2, x_2, x_3)$
- Output: $(y_1, y_2, S(x) \oplus y_1 \oplus y_2)$
- Randomizes the index $a' = a \oplus r \oplus x_2 \oplus x_3$ for $0 \leq a \leq 2^n - 1$
- Shifts the table by y_1, y_2 in one step

Generic 20-secure masking (Prouff-Rivain FSE, 2008)

$$r \in \{0,1\}^n \qquad r' = (r \oplus x_2) \oplus x_3$$

$$y_1, y_2 \in \{0,1\}^n \qquad x = x_1 \oplus x_2 \oplus x_3$$



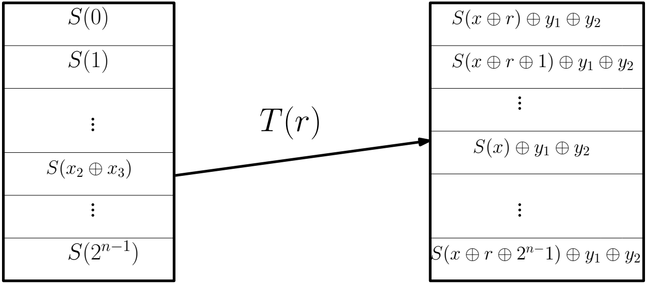
$$T(a') = ((S(x_1 \oplus a) \oplus y_1) \oplus y_2)$$

$$a' = a \oplus r'$$

Generic 20-secure masking (Prouff-Rivain FSE, 2008)

$$r \in \{0,1\}^n \qquad r' = (r \oplus x_2) \oplus x_3$$

$$y_1, y_2 \in \{0,1\}^n \qquad x = x_1 \oplus x_2 \oplus x_3$$



$$a = x_2 \oplus x_3, a' = r$$

$$T(r) = S(x_1 \oplus x_2 \oplus x_3) \oplus y_1 \oplus y_2$$

Generic 2O-secure masking (Prouff-Rivain FSE, 2008)

Algorithm 1 Sec2O-masking

Input: Three input shares: $(x_1 = x \oplus x_2 \oplus x_3, x_2, x_3) \in \mathbb{F}_{2^n}$, two output shares: $(y_1, y_2) \in \mathbb{F}_{2^m}$, and an (n, m) S-box lookup function S

Output: Masked S-box output: $S(x) \oplus y_1 \oplus y_2$

```
r ← Rand(n)
r' ← (r ⊕ x2) ⊕ x3
for a := 0 to 2^n − 1 do
    a' ← a ⊕ r'
    T[a'] ← ((S(x1 ⊕ a) ⊕ y1) ⊕ y2)
end for
return T[r]
```

Vadnala-Großschädl Scheme

- Boolean to arithmetic conversion
  - Input: $x_1 = x \oplus x_2 \oplus x_3, x_2, y_3$
  - Output: $A_1 = x - A_2 - A_3, A_2, A_3$
  - Generate A_2, A_3 randomly
  - Compute $A_1 = x - A_2 - A_3$ using modified LUT

$$T(a') = (x_1 \oplus a) - A_2 - A_3$$

- Arithmetic to Boolean conversion is obtained in the same way

Vadnala-Großschädl Scheme

- Boolean to arithmetic conversion
  - Input: $x_1 = x \oplus x_2 \oplus x_3, x_2, y_3$
  - Output: $A_1 = x - A_2 - A_3, A_2, A_3$
  - Generate A_2, A_3 randomly
  - Compute $A_1 = x - A_2 - A_3$ using modified LUT

$$T(a') = (x_1 \oplus a) - A_2 - A_3$$

- Arithmetic to Boolean conversion is obtained in the same way

Vadnala-Großschädl Scheme

- Boolean to arithmetic conversion
  - Input: $x_1 = x \oplus x_2 \oplus x_3, x_2, y_3$
  - Output: $A_1 = x - A_2 - A_3, A_2, A_3$
  - Generate A_2, A_3 randomly
  - Compute $A_1 = x - A_2 - A_3$ using modified LUT

$$T(a') = (x_1 \oplus a) - A_2 - A_3$$

- Arithmetic to Boolean conversion is obtained in the same way

Vadnala-Großschädl Scheme

- Boolean to arithmetic conversion
  - Input: $x_1 = x \oplus x_2 \oplus x_3, x_2, y_3$
  - Output: $A_1 = x - A_2 - A_3, A_2, A_3$
  - Generate A_2, A_3 randomly
  - Compute $A_1 = x - A_2 - A_3$ using modified LUT

$$T(a') = (x_1 \oplus a) - A_2 - A_3$$

- Arithmetic to Boolean conversion is obtained in the same way

Improved B→A conversion algorithm

- Use divide-and-conquer
- Divide each share into p parts of l bits each; $n = p \cdot l$
- Convert each part separately using previous approach
- Problem: Carries

Improved B→A conversion algorithm

- Use divide-and-conquer
- Divide each share into p parts of l bits each; $n = p \cdot l$
- Convert each part separately using previous approach
- Problem: Carries

Improved B→A conversion algorithm

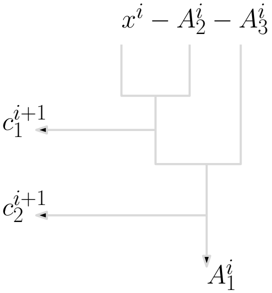
- Use divide-and-conquer
- Divide each share into p parts of l bits each; $n = p \cdot l$
- Convert each part separately using previous approach
- Problem: Carries

Improved B→A conversion algorithm

- Use divide-and-conquer
- Divide each share into p parts of l bits each; $n = p \cdot l$
- Convert each part separately using previous approach
- Problem: Carries

Handling carries

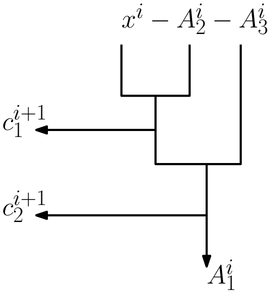
- Original equation: $(A_1)^i = x^i - A_2 - A_3$ (The subtraction here are performed modulo 2^l instead of 2^n)



- New equation: $(A_1)^i = x^i - c_1^i - A_2 - c_2^i - A_3$
- Output carries of word $i \implies$ Input carries of word $i+1$

Handling carries

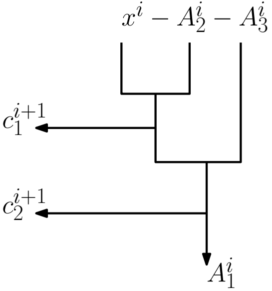
- Original equation: $(A_1)^i = x^i - A_2 - A_3$ (The subtraction here are performed modulo 2^l instead of 2^n)



- New equation: $(A_1)^i = x^i - c_1^i - A_2 - c_2^i - A_3$
- Output carries of word $i \implies$ Input carries of word $i+1$

Handling carries

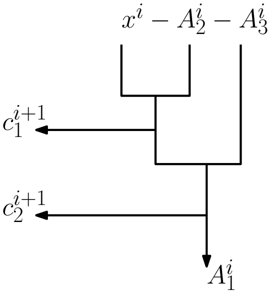
- Original equation: $(A_1)^i = x^i - A_2 - A_3$ (The subtraction here are performed modulo 2^l instead of 2^n)



- New equation: $(A_1)^i = x^i - c_1^i - A_2 - c_2^i - A_3$
- Output carries of word $i \implies$ Input carries of word $i+1$

Handling carries

- Original equation: $(A_1)^i = x^i - A_2 - A_3$ (The subtraction here are performed modulo 2^l instead of 2^n)



- New equation: $(A_1)^i = x^i - c_1^i - A_2 - c_2^i - A_3$
- Output carries of word $i \implies$ Input carries of word $i+1$

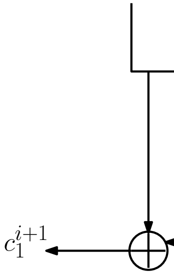
Computing carries

- New equation: $(A_1)^i = x^i - c_1^i - A_2 - c_2^i - A_3$

$$(A_1)^i = \boxed{x^i - c_1^i - A_2} - c_2^i - A_3$$

$$\downarrow$$

$$x^i - c_1^i - A_2$$



Protecting carries

- Problem: Carries can still leak
- *Solution:* Apply generic countermeasure again
- Total of three LUTs

$$\begin{aligned} T_1 &: 2^{l+2} \cdot l \quad (A_1^i) \\ T_2 &: 2^{l+2} \cdot 1 \quad (c_1^{i+1}) \\ T_3 &: 2^{l+2} \cdot 1 \quad (c_2^{i+1}) \end{aligned}$$

- Complexity: $\mathcal{O}(2^{l+2} \cdot p)$ (Earlier scheme: $\mathcal{O}(2^{l \cdot p})$)

Protecting carries

- Problem: Carries can still leak
- *Solution:* Apply generic countermeasure again
- Total of three LUTs

$$\begin{aligned} T_1 &: 2^{l+2} \cdot l \quad (A_1^i) \\ T_2 &: 2^{l+2} \cdot 1 \quad (c_1^{i+1}) \\ T_3 &: 2^{l+2} \cdot 1 \quad (c_2^{i+1}) \end{aligned}$$

- Complexity: $\mathcal{O}(2^{l+2} \cdot p)$ (Earlier scheme: $\mathcal{O}(2^{l \cdot p})$)

Protecting carries

- Problem: Carries can still leak
- *Solution:* Apply generic countermeasure again
- Total of three LUTs

$$\begin{aligned} T_1 &: 2^{l+2} \cdot l \quad (A_1^i) \\ T_2 &: 2^{l+2} \cdot 1 \quad (c_1^{i+1}) \\ T_3 &: 2^{l+2} \cdot 1 \quad (c_2^{i+1}) \end{aligned}$$

- Complexity: $\mathcal{O}(2^{l+2} \cdot p)$ (Earlier scheme: $\mathcal{O}(2^{l \cdot p})$)

Protecting carries

- Problem: Carries can still leak
- *Solution:* Apply generic countermeasure again
- Total of three LUTs

$$\begin{array}{lcll} T_1 & : & 2^{l+2} \cdot l & (A_1^i) \\ T_2 & : & 2^{l+2} \cdot 1 & (c_1^{i+1}) \\ T_3 & : & 2^{l+2} \cdot 1 & (c_2^{i+1}) \end{array}$$

- Complexity: $\mathcal{O}(2^{l+2} \cdot p)$ (Earlier scheme: $\mathcal{O}(2^{l \cdot p})$)

Security Analysis

- For securing one word: Similar to Prouff-Rivian
- Every pair is independent of the sensitive variable
- For the full algorithm: Mathematical induction

Security Analysis

- For securing one word: Similar to Prouff-Rivian
- Every pair is independent of the sensitive variable
- For the full algorithm: Mathematical induction

Security Analysis

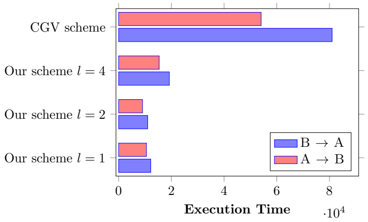
- For securing one word: Similar to Prouff-Rivian
- Every pair is independent of the sensitive variable
- For the full algorithm: Mathematical induction

Implementation results

Algorithm	ℓ	Time	Memory	rand
second-order conversion				
Algorithm B→A	1	12186	8	226
Algorithm B→A	2	11030	16	114
Algorithm B→A	4	19244	64	58
Algorithm A→B	1	10557	8	226
Algorithm A→B	2	9059	16	114
Algorithm A→B	4	15370	64	58
CGV $A \to B$	-	54060	-	484
CGV $B \to A$	-	81005	-	822
first-order addition				
KRJ addition	-	371	-	1
Our algorithm	4	294	512	3

Table : Implementation results for $n = 32$ on a 32-bit microcontroller.

Implementation results

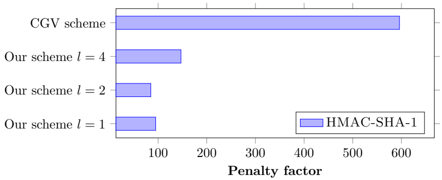


Application to HMAC-SHA-1

Algorithm	ℓ	Time	PF
HMAC-SHA-1	-	104	1
second-order conversion			
Our solution	1	9715	95
Our solution	2	8917	85
Our solution	4	15329	147
CGV	-	62051	596
first-order addition			
KRJ addition	-	328	3.1
Our solution	4	308	2.9

Table : Running time in thousands of clock cycles and penalty factor compared to the unmasked HMAC-SHA-1 implementation

Application to HMAC-SHA-1



Conclusions

- Improved algorithms for second-order conversion
- Requires only 3 shares and works for larger conversion size
- First-order masked addition using LUT
- Significant improvement (85%)in execution time for second-order conversion
- Alternative solution for first-order addition

Conclusions

- Improved algorithms for second-order conversion
- Requires only 3 shares and works for larger conversion size
- First-order masked addition using LUT
- Significant improvement (85%)in execution time for second-order conversion
- Alternative solution for first-order addition

Conclusions

- Improved algorithms for second-order conversion
- Requires only 3 shares and works for larger conversion size
- First-order masked addition using LUT
- Significant improvement (85%)in execution time for second-order conversion
- Alternative solution for first-order addition

Conclusions

- Improved algorithms for second-order conversion
- Requires only 3 shares and works for larger conversion size
- First-order masked addition using LUT
- Significant improvement (85%)in execution time for second-order conversion
- Alternative solution for first-order addition

Conclusions

- Improved algorithms for second-order conversion
- Requires only 3 shares and works for larger conversion size
- First-order masked addition using LUT
- Significant improvement (85%)in execution time for second-order conversion
- Alternative solution for first-order addition