Suwako: A Logarithmic-Depth Modular Reduction for Arbitrary Trinomials over $\mathbb{F}_{2^m}$ without Pre-computation

Junyu Zhou[1], Jing Wang[2], Hao Ren[3], Si Gao[4], and Xiao Lan[3]

[1] Cyber Science Research Institute, Sichuan University, Sichuan, China
helloworld@junyu33.me
[2] School of Integrated Circuits, Huazhong University of Science and Technology, Wuhan, China
nick585108@163.com
[3] Cyber Science Research Institute, Sichuan University, Sichuan, China
{hao.ren,lanxiao}@scu.edu.cn
[4] Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences
gaosi@iscas.ac.cn

Abstract. Modular reduction over binary extension fields $\mathbb{F}_{2^m}$ is a fundamental operation in cryptographic implementations, including GCM and Elliptic Curve Cryptography. Traditional reduction algorithms (e.g., linear LFSR-based methods) are highly sensitive to the algebraic structure of the defining polynomial. This sensitivity is especially acute for trinomials $P(x) = x^m + x^t + 1$, where cryptographic standards have historically mandated the use of "friendly" polynomials (with small t) to avoid the linear performance degradation associated with "random" or "unfriendly" parameters. In this paper, we challenge this constraint by introducing Suwako, a novel reduction algorithm. By exploiting the self-similar algebraic structure of the reduction map, Suwako transforms the reduction process from a serial iterative chain (dependent on the degree gap $\Delta = m - t$) into a logarithmic-depth binary-doubling structure. We theoretically prove that Suwako achieves $O(\log m)$ folding depth for arbitrary trinomials, regardless of the position of the middle term t. Furthermore, unlike window-based or Montgomery/Barrett reduction methods, Suwako requires no pre-computation, making it optimal for dynamic environments.

Keywords: $\mathbb{F}_{2^m}$ Arithmetic · Modular Reduction · Trinomials · Logarithmic Complexity

1 Introduction

Finite field arithmetic, particularly over binary extension fields $\mathbb{F}_{2^m}$, serves as the computational backbone for modern cryptographic protocols, ranging from traditional Elliptic Curve Cryptography (ECC) [Mil86] to post-quantum candidates like Hamming-Quasi-Cyclic (HQC) codes [AMAB+19]. Performance profiling indicates that arithmetic operations in $\mathbb{F}_{2^m}$ often consume 53% to 97% of

the total runtime in these applications, making their optimization critical. Over the past decades, significant progress has been made in optimizing polynomial multiplication. The literature is rich with advanced techniques, including Karatsuba and its variants [KO62,Too63,CA69,GK10], Fast Fourier Transform (FFT) or Number Theoretic Transform (NTT) approaches [SS71,Pol71,HHL17,TH22], and various optimizations ranging from software libraries [BGTZ08], specific instruction set extensions (e.g., `PCLMULQDQ` on x86 [Int14,Int23,WMK16] and `Zbc` on RISC-V [RIS22]), to GCM-specific implementations [GL15,RV23].

However, multiplication is only half the story. The subsequent step, modular reduction, remains a persistent bottleneck, especially for constrained environments. While general-purpose reduction algorithms such as Montgomery [Mon85], Barrett [Bar86], and their modern variants [LN16,Pla21] effectively convert reduction into multiplication, they suffer from a fundamental limitation: they require the pre-computation of constants tied to a specific modulus. This pre-computation prevents parameter agility, making them unsuitable for dynamic environments. Consequently, cryptographic standards often resort to "friendly" irreducible polynomials (e.g., trinomials x^m+x^t+1 with small t) to allow for efficient linear reduction (folding-based methods). This restriction limits the choice of field parameters, as "unfriendly" trinomials (where $t > m/2$) exhibit poor performance with traditional linear reduction algorithms.

To address this gap, we introduce **Suwako**, a novel modular reduction algorithm designed for arbitrary trinomials over $\mathbb{F}_{2^m}$. Unlike traditional methods that treat the reduction process as a linear chain of shifts and XORs, Suwako exploits the self-similar algebraic structure of trinomials to transform the reduction into a binary-doubling process. We provide a Python-based proof-of-concept implementation [5] to validate the efficiency of our reduction. On random trinomials over $\mathbb{F}_{2^{2048}}$, our method demonstrates an approximate $50\times$ speedup compared to the standard bit-level reduction baseline.

Contribution: We propose the first software feasible reduction algorithm that achieves a **logarithmic structural depth** of $O(\log m)$ for **any** trinomial $x^m + x^t + 1$, regardless of the position of the middle term t. Traditional reduction methods typically exhibit a linear complexity $O(m/\Delta)$ dependent on the degree gap $\Delta = m - t$, which degrades significantly when t is large. In contrast, Suwako decouples the computational complexity from the polynomial structure. By recursively "folding" the high-degree terms using a doubling strategy, our method effectively "jumps" over the linear reduction steps. Crucially, this approach requires no pre-computation, distinguishing it from Montgomery/Barrett-like methods.

2 Preliminaries

2.1 Complexity Metrics

We evaluate our reduction algorithms using the word model. Unless otherwise specified, we assume a machine word size of $W = 64$ bits. The metrics include:

[5] The implementation is available at https://github.com/junyu33/suwako.

- **Time Complexity (T):** The total number of W-bit logical operations (XORs and shifts) required to complete the reduction.
- **Folding Depth (D):** The number of recursive steps or iterations performed. This reflects the algorithm's control-flow overhead and its suitability for SIMD vectorization, where fewer rounds with higher parallelism are preferred.

2.2 Notations

Table 1 summarizes the key notations used in this work.

Table 1: Key notations and their meanings

Notation	Meaning
$\times$	Multiplication.
$\%$	The modulo operation.
$\&, \vert, \oplus$	Bitwise AND, OR and XOR.
$\ll, \gg$	Bitwise left-shift, right-shift operation.
m	The degree of the finite field being calculated.
W	The width of machine, typically 64, but up to 512 using SIMD.
w_h	The hamming weight (terms) of the irreducible polynomial.
$H_{[a,b]}$	The bit slicing of polynomial H between lowest a bit and b bit.
$H_{[a]}$	The only bit a of polynomial H.
$[a:b]$	Integer concatenation of a and b.
$\lfloor x \rfloor, \lceil x \rceil$	Round down, round up of x.
$g(x)$	Irreducible polynomial of $\mathbb{F}_{2^m}$.
$M(m)$	The time complexity of multiplying m bit polynomial.
$\deg_n(g)$	The n-th largest exponent of the polynomial $g(x)$.
Δ	$m - \deg_2(p)$.

3 Analysis on Related Works

Modular reduction in $\mathbb{F}_{2^m}$ underpins core cryptographic primitives, with algorithms trading off between asymptotic complexity, modulus sparsity, and precomputation requirements. Table 2 summarizes the landscape.

Bit-Level and Folding Strategies. The baseline **Long Division** iteratively cancels the leading term of the polynomial. While conceptually simple and universally applicable, it requires $O(m)$ iterations, yielding a quadratic cost $O(m^2/W)$. The **XOR-Fold** method improves upon this by exploiting the modulus structure $x^m \equiv \sum_{j \in J} x^j$, where $g(x) = x^m + \sum_{j \in J} x^j$. By reducing Δ bits per step using only shifts and XORs, it lowers the complexity to $O(m^2 w_h/(W\Delta))$. This is particularly effective for sparse polynomials, but performance degrades significantly when the gap Δ is small or the Hamming weight w_h is large.

Table 2: Comparison of $\mathbb{F}_{2^m}$ Software Modular Reduction Algorithms

	Long division	**Xor-Fold [LD00]**	**[GK10]**	**Ad-hoc**	**Our work**
T_{sparse}	$O(\frac{m^2}{W})$	$O(\frac{m^2 w_h}{W\Delta})$	$O(\frac{mw_h}{W})$ $O(M(m))$	$O(\frac{mw_h}{W})$	$O(\frac{m}{W}\log m)$
T_{dense}	$O(\frac{m^2}{W})$	$O(\frac{m^3}{W\Delta})$	$O(\frac{m^2}{W^2})$ $O(M(m))$	✗	✗
D	$O(m)$	$O(\frac{m}{W})$	$O(\frac{m}{W})$ 2	2	$O(\log m)$
Generality	✓	✓	$\Delta \geq W$ ✓	✗	trinomials
No precomp	✓	✓	✓ ✗	✓	✓

Word-Level and Ad-hoc Methods. To leverage modern CPU word sizes, **López and Dahab [LD00]** proposed a serial shift-and-XOR mechanism that processes data in W-bit blocks. This approach is highly efficient for sparse polynomials, achieving $O(mw_h/W)$ complexity. However, it suffers from a linear dependency chain of depth $O(m/W)$ and fails when the polynomial gap Δ<W, requiring expensive bit-level adjustments. **Ad-hoc** methods [HMV10] optimize this further by hardcoding the shift sequences for a specific fixed modulus (e.g., NIST curves), achieving optimal speed at the cost of losing all parameter agility.

Multiplication-Based Reduction. For dense polynomials or architectures with fast carry-less multiplication (`CLMUL`), **Gueron [GK10]** adapted Barrett reduction to binary fields. By precomputing constants derived from $g(x)$, reduction is transformed into two multiplications and truncations, with cost $O(M(m))$ independent of sparsity. While offering constant-time properties for dense moduli, the requirement for pre-computation hinders agility in scenarios where parameters change dynamically.

4 Log-Depth Modular Reduction for Trinomials

Existing reduction schemes for trinomials are fundamentally constrained by their linear sequential structure. Traditional folding-based methods and word-level reduction require a folding depth of $O(m/\Delta)$. Although [GK10] and [HMV10] have a folding depth of 2, they require precomputed constants or designated irreducible trinomials.

In contrast, our proposed algorithm, Suwako, achieves a logarithmic folding depth of $O(\log(m/\Delta))$. By leveraging congruence relations, Suwako reshapes the reduction process into a binary doubling strategy, performing total computational work of $O(\frac{m}{W}\log\frac{m}{\Delta})$. The significance of this approach lies in the structural elimination of the linear-depth bottleneck: Suwako breaks the long dependency chains inherent in bitwise or word-level serial methods, enabling SIMD operations in software and low latency in hardware implementation.

To realize this improvement, we exploit a bit-parallel divide-and-XOR strategy that generalizes the well-known "parallel prefix XOR" pattern from low-level bit manipulation. The key observation is that repeated folding of overlapping or fixed-size Δ-bit blocks can be performed hierarchically in a binary-doubling

manner, rather than through sequential word-by-word propagation. Specifically, Lemma 1 and Lemma 2 formalize two dual variants of this procedure: the former computes cumulative XORs of variable-length suffixes, while the latter collapses all fixed-width Δ-blocks into a single Δ-bit word. Both schemes require only $O(\log(m/\Delta))$ rounds of shift–XOR operations. This approach is conceptually related to the XOR folding idiom described in *Hacker's Delight* [HSW13] and some applications in [KRS85,HS86], but to our knowledge, its binary-doubling formulation and formal correctness proof have not been explicitly stated in prior arithmetic or reduction literature.

Lemma 1 (suffix XOR). *Let $H \in \{0,1\}^m$ be viewed as a nonnegative integer (least significant bit at position 0), let $\Delta \in \mathbb{N}$, $\Delta \geq 1$, and set $L := \left\lfloor \frac{m-1}{\Delta} \right\rfloor$. Define the right-shift $\gg$ as logical (zero-filling). Then*

$$\bigoplus_{i=1}^{L} H_{[i\Delta,\, m-1]} \;=\; \bigoplus_{i=1}^{L} (H \gg i\Delta), \tag{1}$$

and the right-hand side can be computed in $O(\log L)$ shifts and XORs by the binary-doubling scheme

$$T^{(0)} := H \gg \Delta, \qquad T^{(k+1)} := T^{(k)} \oplus \left(T^{(k)} \gg \Delta\, 2^k\right) \quad (k \geq 0), \tag{2}$$

terminating after $r := \lceil \log_2 L \rceil$ iterations, with $T^{(r)} = \bigoplus_{i=1}^{L} (H \gg i\Delta)$.

Proof. First, $H_{[i\Delta,\, m-1]}$ is exactly $(H \gg i\Delta)$ under the zero-filling convention, hence the stated equality is tautological. It remains to prove the binary-doubling correctness. We claim by induction on $k \geq 0$ that

$$T^{(k)} \;=\; \bigoplus_{i=1}^{\min(L,\, 2^k)} (H \gg i\Delta). \tag{3}$$

For $k = 0$, $T^{(0)} = H \gg \Delta$ matches the claim. Suppose the claim holds for some k. Then

$$T^{(k+1)} \;=\; T^{(k)} \oplus \left(T^{(k)} \gg \Delta\, 2^k\right) \;=\; \bigoplus_{i=1}^{\min(L,\, 2^k)} (H \gg i\Delta) \oplus \bigoplus_{i=1}^{\min(L,\, 2^k)} (H \gg (i+2^k)\Delta). \tag{4}$$

Since terms with $(i + 2^k) > L$ vanish (logical shifts beyond bit $m-1$ produce zeros), the union of the two XOR ranges is precisely $i = 1, \ldots, \min(L,\, 2^{k+1})$, proving the step. Taking $r := \lceil \log_2 L \rceil$ yields $2^r \geq L$, whence $T^{(r)} = \bigoplus_{i=1}^{L} (H \gg i\Delta)$ as claimed. The operation count is $r = O(\log L)$ shifts/XORs. Edge case $L = 0$ gives the empty XOR 0 and vacuous termination. □

Lemma 2 (blockwise Δ-XOR). *Let $H \in \{0,1\}^m$ be viewed as a nonnegative integer with least significant bit at position 0. Fix a block width $\Delta \geq 1$, and let $N := \left\lfloor \frac{m-1}{\Delta} \right\rfloor + 1$ be the number of contiguous Δ-bit blocks in H. Define the*

zero-filling right-shift operator $S^{\alpha}(x) := x \gg \alpha$. *Let* $r := \lceil \log_2 N \rceil$. *Consider the in-place descending scheme*

$$X \leftarrow H; \qquad \textit{for } k = r-1, r-2, \ldots, 0: \quad X \leftarrow X \ \oplus \ S^{\Delta 2^k}(X).$$

Then, after termination, the lowest Δ *bits of* X *equal* $\bigoplus_{i=0}^{N-1} H_{[i\Delta,\, i\Delta+\Delta-1]}$. *Equivalently,* $X \bmod 2^{\Delta} = \bigoplus_{i=0}^{N-1} \big((H \gg i\Delta) \bmod 2^{\Delta}\big)$. *The procedure uses* $r = O(\log N)$ *shifts and XORs.*

Proof. For each k, the update $X \leftarrow X \oplus S^{\Delta 2^k}(X)$ applies the linear operator $I \oplus S^{\Delta 2^k}$ to the current word (I is the identity operator). Since $S^{\alpha} S^{\beta} = S^{\alpha+\beta} = S^{\beta} S^{\alpha}$, we have $(I \oplus S^{\alpha})(I \oplus S^{\beta}) = (I \oplus S^{\beta})(I \oplus S^{\alpha})$. Hence the operators $\{I \oplus S^{\Delta 2^k}\}_{k=0}^{r-1}$ commute, and the descending loop realizes the product

$$\mathcal{P}_r \ := \ \prod_{k=0}^{r-1} \big(I \oplus S^{\Delta 2^k}\big) \ = \ \bigoplus_{j=0}^{2^r-1} S^{\Delta j}, \tag{5}$$

where the equality follows by expanding the product over $\mathbb{F}_2$ (each subset of $\{0, \ldots, r-1\}$ contributes one shift by the sum of its chosen exponents, and every $j \in \{0, \ldots, 2^r - 1\}$ has a unique binary decomposition). Applying $\mathcal{P}_r$ to H yields

$$X \ = \ \mathcal{P}_r(H) \ = \ \bigoplus_{j=0}^{2^r-1} S^{\Delta j}(H) \ = \ \bigoplus_{j=0}^{N-1} (H \gg j\Delta), \tag{6}$$

because $2^r \geq N$ and terms with $j \geq N$ vanish under zero-fill shifting (they shift all Δ-blocks past bit $m-1$). Finally, taking the lowest Δ bits extracts, for each summand $(H \gg j\Delta)$, exactly the block $H_{[j\Delta,\, j\Delta+\Delta-1]}$ aligned at positions $[0, \Delta-1]$. Therefore $X \bmod 2^{\Delta} = \bigoplus_{j=0}^{N-1} H_{[j\Delta,\, j\Delta+\Delta-1]}$. This proves correctness; the loop count is $r = O(\log N)$. □

Building upon the preceding lemmas for suffix and blockwise XOR folding, we now instantiate the reduction procedure for trinomials. Assume that the modulus is an irreducible trinomial $g(x) = x^m + x^t + 1$. Let C denote the $2m$-bit carry-less multiplication result. We set $\Delta = m - t$. Based on the derived expressions for the two folded XOR terms, the complete reduction procedure is summarized in Algorithm 1 , which applies to arbitrary trinomials and exploits the binary-doubling structure for efficient in-place computation.

Theorem 1. *Algorithm 1 is correct.*

Proof. We split $2m$-bit C into m-bit L and H satisfying $C = L \oplus H \times x^m$, and denote by $C_{[a,b]}$ the bit slice of C from position a (least significant) to position b; for example, if $C = 0b11100110$, then $C_{[2,5]} = 0b1001$. We focus on computing the intermediate value $res = (H \times x^m) \bmod g(x) = (H \times x^m) \bmod (x^m + x^t + 1)$, since the final result is obtained simply by XORing res with L.

Algorithm 1: Suwako Reduction over $g(x) = x^m + x^t + 1$

Input: Integers $m \geq 1$, $t \in (0, m)$ with $g(x) = x^m + x^t + 1$ irreducible; $C \in \{0,1\}^{2m}$.
Output: $R = C \bmod g \in \{0,1\}^m$.

Preliminaries: $\Delta \leftarrow m - t$; $H \leftarrow C \gg m$; $L \leftarrow C \;\&\; (2^m - 1)$;

Fold #1 (suffix XOR): $N \leftarrow \lfloor \frac{m-1}{\Delta} \rfloor$;
if $N = 0$ **then**
  $S_1 \leftarrow 0$
else
  $T \leftarrow H \gg \Delta$;
  for $k \leftarrow 0$ **to** $\lceil \log_2 N \rceil - 1$ **do**
    $T \leftarrow T \;\oplus\; \big(T \gg (\Delta \cdot 2^k)\big)$
  $S_1 \leftarrow T$;

Fold #2 (blockwise Δ-XOR): $N \leftarrow \lfloor \frac{m-1}{\Delta} \rfloor + 1$; $r \leftarrow \lceil \log_2 N \rceil$; $X \leftarrow H$;
for $k \leftarrow r - 1$; $k \geq 0$; $k \leftarrow k - 1$ **do**
  $X \leftarrow X \;\oplus\; \big(X \gg (\Delta \cdot 2^k)\big)$
$S_2 \leftarrow X \;\&\; (2^\Delta - 1)$;

Assemble and Return: $A \leftarrow S_1 \ll \Delta$; $\quad B \leftarrow (S_2 \ll t) \;\oplus\; S_2$;
$R \leftarrow A \;\oplus\; B \;\oplus\; L$; **return** R;

$$
\begin{aligned}
res &= (H \times x^m) \% g && (7)\\
&= H \times (x^t \oplus 1) \% g && (8)\\
&= (H_{[0,\Delta-1]} \oplus x^\Delta H_{[\Delta,m-1]}) \times (x^t \oplus 1) \% g && (9)\\
&= (x^t \oplus 1)(x^\Delta H_{[\Delta,m-1]}) \% g \oplus H_{[0,\Delta-1]}(x^t \oplus 1) \% g && (10)\\
&= (H_{[\Delta,m-1]} \times x^m) \% g \oplus && (11)\\
&\quad (x^\Delta H_{[\Delta,m-1]}) \% g \oplus H_{[0,\Delta-1]}(x^t \oplus 1) \% g && (12)
\end{aligned}
$$

It is easy to observe that equation 11 is a subproblem of $(H \times x^m) \% g$ itself, and the degree of two parts of equation 12 are less then m, then we can safely continue the recursion:

$$
\begin{aligned}
res &= (H_{[\Delta,m-1]} \times x^m) \% g \oplus\\
&\quad (x^\Delta H_{[\Delta,m-1]}) \% g \oplus H_{[0,\Delta-1]}(x^t \oplus 1) \% g && (13)\\
&= (H_{[2\Delta,m-1]} \times x^m) \% g \oplus\\
&\quad (x^\Delta H_{[2\Delta,m-1]}) \% g \oplus H_{[\Delta,2\Delta-1]}(x^t \oplus 1) \% g \oplus\\
&\quad (x^\Delta H_{[\Delta,m-1]}) \% g \oplus H_{[0,\Delta-1]}(x^t \oplus 1) \% g && (14)\\
&= \cdots\\
&= (x^\Delta H_{[\lfloor \frac{m-1}{\Delta} \rfloor \Delta,m-1]}) \% g \oplus H_{[\lfloor \frac{m-1}{\Delta} \rfloor \Delta,m-1]}(x^t \oplus 1) \% g \oplus
\end{aligned}
$$

$$
\begin{aligned}
&\cdots \\
&(x^{\Delta} H_{[2\Delta,m-1]})\%g \oplus H_{[\Delta,2\Delta-1]}(x^t \oplus 1)\%g \oplus \\
&(x^{\Delta} H_{[\Delta,m-1]})\%g \oplus H_{[0,\Delta-1]}(x^t \oplus 1)\%g \qquad (15) \\
&= x^{\Delta} \bigoplus_{i=1}^{\lfloor \frac{m-1}{\Delta} \rfloor} H_{[i\Delta,m-1]}\%g \oplus (x^t \oplus 1) \bigoplus_{i=0}^{\lfloor \frac{m-1}{\Delta} \rfloor} H_{[i\Delta,i\Delta+\Delta-1]}\%g \qquad (16)
\end{aligned}
$$

By repeatedly applying these congruence relations, finally we have made each term's degree less than m, now the reduction problem transltes to how to quickly calculate

$$
\bigoplus_{i=1}^{\lfloor \frac{m-1}{\Delta} \rfloor} H_{[i\Delta,m-1]} \quad \text{and} \quad \bigoplus_{i=0}^{\lfloor \frac{m-1}{\Delta} \rfloor} H_{[i\Delta,i\Delta+\Delta-1]}.
$$

Lemma 1 and Lemma 2 already demonstrate that the folding phase of the reduction — i.e., summing all relevant terms — has a logarithmic control depth of $O\big(\log\lfloor \frac{m-1}{\Delta} \rfloor\big)$. However, each folding layer requires a full linear pass over the operand, leading to an overall work complexity of $O\big(\frac{m}{W} \log \frac{m}{\Delta}\big)$. The remaining steps involve the multiplications by x^{Δ} and x^t, which are realized as simple bit-shift operations. These shifts contribute a single additional linear pass with cost $O\big(\frac{m}{W}\big)$, which does not change the asymptotic complexity. □

5 Conclusion and Future Work

In this paper, we presented Suwako, an algorithm that fundamentally transforms modular reduction over $\mathbb{F}_{2^m}$ from a serial linear process into a parallelizable logarithmic-depth operation $O(\log m)$ for any trinomial. By overcoming the serial dependency bottlenecks inherent in traditional folding methods, Suwako offers a unified, parameter-agnostic structure that maintains high efficiency and regularity even for challenging trinomials where $\Delta < W$. Future research will focus on two promising directions:

- **High-Performance Hardware:** Suwako's low-depth topology naturally supports high-frequency pipelining, making it an ideal candidate for next-generation accelerators in Post-Quantum Cryptography (PQC) and Zero-Knowledge Proofs (ZKPs), where large-field arithmetic performance is critical.
- **Privacy-Preserving Protocols:** In the context of Secure Multi-Party Computation (MPC), the multiplicative depth of a circuit dictates the round complexity. Suwako's logarithmic depth theoretically allows for *oblivious reduction* with significantly reduced communication rounds compared to linear methods, offering a potential breakthrough for low-latency privacy-preserving applications.

Acknowledgements

Origin of the Name: The algorithm is named Suwako after *Suwako Moriya* from the *Touhou Project* series. In the lore, Suwako is a native god capable of creating "Earth" (*Kuniumi*). This serves as a metaphor for the algorithm's manipulation of the Finite Field, drawing a parallel between the goddess's dominion over the land and the mathematical structure. Additionally, reflecting her affinity with frogs, the algorithm mirrors her agile leaping capabilities, symbolizing how our doubling strategy "jumps" over the traditional linear dependency chain to achieve logarithmic speed.

References

[AMAB+19] Carlos Aguilar-Melchior, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor. Hamming Quasi-Cyclic (HQC). In *NIST Post-Quantum Cryptography submissions, round 2*. NIST, 2019. Accessed: April 06, 2025.

[Bar86] Paul Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 311–323. Springer, 1986.

[BGTZ08] Richard P. Brent, Pierrick Gaudry, Emmanuel Thomé, and Paul Zimmermann. Faster Multiplication in gf(2)[x]. In *Algorithmic Number Theory: 8th International Symposium, ANTS-VIII*, volume 5011 of *Lecture Notes in Computer Science*, pages 153–166, Berlin, Heidelberg, 2008. Springer.

[CA69] Stephen A. Cook and Stål O. Aanderaa. On the Minimum Computation Time of Functions. *Transactions of the American Mathematical Society*, 142:291–314, 1969.

[GK10] Shay Gueron and Michael E. Kounavis. Efficient Implementation of the Galois Counter Mode using a Carry-less Multiplier and a Fast Reduction Algorithm. *Information Processing Letters*, 110(14-15):549–553, 2010.

[GL15] Conrado P. L. Gouvêa and Julio López. Implementing GCM on ARMv8. In *Topics in Cryptology – CT-RSA 2015*, volume 9048 of *Lecture Notes in Computer Science*, pages 167–180. Springer, Cham, 2015.

[HHL17] David Harvey, Joris van der Hoeven, and Grégoire Lecerf. Faster Polynomial Multiplication over Finite Fields. *Journal of the ACM*, 63(6), January 2017.

[HMV10] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[HS86] W. Daniel Hillis and Guy L. Steele. Data Parallel Algorithms. *Commun. ACM*, 29(12):11701183, December 1986.

[HSW13] Jr. Henry S. Warren. *Hacker's Delight*. Addison-Wesley, 2nd edition, 2013.

[Int14] Intel Corporation. Carry-less Multiplication and Its Usage for Computing the GCM Mode. Technical Report Rev. 2.02, Intel Corporation, April 2014.

[Int23] Intel Corporation. Advanced Encryption Standard Galois Counter Mode – Optimized GHASH Function Technology Guide. Technical report, Intel Corporation, August 2023.

[KO62] Anatolii Karatsuba and Yuri Ofman. Multiplication of Multidigit Numbers on Automata. *Soviet Physics Doklady*, 7:595, 12 1962.

[KRS85] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. The Power of Parallel Prefix. *IEEE Transactions on Computers*, C-34(10):965–968, 1985.

[LD00] Julio López and Ricardo Dahab. High-Speed Software Multiplication in $\mathbb{F}_2^m$. In *Proceedings of the First International Conference on Progress in Cryptology*, INDOCRYPT '00, page 203212, Berlin, Heidelberg, 2000. Springer-Verlag.

[LN16] Patrick Longa and Michael Naehrig. Speeding Up the Number Theoretic Transform for Faster Ideal Lattice-based Cryptography. In *International Conference on Cryptology and Network Security*, pages 124–139. Springer, 2016.

[Mil86] Victor S. Miller. Use of Elliptic Curves in Cryptography. In *Advances in Cryptology – CRYPTO '85 Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, Berlin, Heidelberg, 1986.

[Mon85] Peter L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985.

[Pla21] Thomas Plantard. Efficient Word Size Modular Arithmetic. *IEEE Transactions on Emerging Topics in Computing*, 9(3):1506–1518, 2021.

[Pol71] John M Pollard. The Fast Fourier Transform in a Finite Field. *Mathematics of computation*, 25(114):365–374, 1971.

[RIS22] RISC-V International. RISC-V Cryptographic Instruction Set Extensions: Vector Extra Instructions. Technical Specification Draft 0.0.5, RISC-V International, August 2022.

[RV23] Jean-Marc Robert and Pascal Véron. Faster Multiplication over $\mathbb{F}_2[x]$ using AVX512 Instruction Set and VPCLMULQDQ Instruction. *Journal of Cryptographic Engineering*, 13(1):37–55, 2023.

[SS71] Arnold Schönhage and Volker Strassen. Schnelle Multiplikation Großer Zahlen. *Computing*, 7:281–292, 1971.

[TH22] Nianqi Tang and Yunghsiang S. Han. A New Decoding Method for Reed-Solomon Codes Based on FFT and Modular Approach. *IEEE Transactions on Communications*, 70(12):7790–7801, 2022.

[Too63] Andrei L Toom. The Complexity of a Scheme of Functional Elements Simulating the Multiplication of Integers. In *Doklady Akademii Nauk*, volume 150, pages 496–498. Russian Academy of Sciences, 1963.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient Multi-Party Computation Toolkit. https://github.com/emp-toolkit, 2016.