

Area-Time Efficient NTRU Prime Decapsulation: ASIC Evaluation of the First Five-Way Char-3 Multiplier

Esra Yeniaras*

Abstract

Streamlined NTRU Prime (**sntrup761**) is a lattice-based key encapsulation mechanism that, although not a NIST standard, remains widely deployed in critical internet infrastructure. It is the post-quantum key-exchange default in OpenSSH, standardized in RFC 9941, and used well beyond SSH, in Red Hat Enterprise Linux, the liboqs library, PQConnect, and commercial VPNs. Its decapsulation performs a polynomial multiplication over the characteristic-three ring $\mathbb{Z}_3[x]/(x^p - x - 1)$, so faster methods for this operation directly improve these protocols. We use the Yeniaras-Cenk 5-way multiplier (U1-hybrid), which has the lowest arithmetic complexity among characteristic-three multipliers and gives a 35.52% scalar-C software speedup over Bernstein’s three-way method (B1). Yet every prior NTRU Prime hardware design, on FPGA or the single existing ASIC, uses only schoolbook or a single layer of two-way Karatsuba. Three-way splits have been implemented only in software; no 5-way split had been implemented at all before this work.

We present the first ASIC evaluation of the Yeniaras-Cenk 5-way multiplier (U1-hybrid), synthesized to the Nangate 45 nm library and compared against the state-of-the-art parallel schoolbook multiplier of Peng et al. On the area-delay product (ADP), the Yeniaras-Cenk 5-way multiplier is $6.8\times$ better than the Peng schoolbook at the multiplier level, and $1.27\times$ better in the **sntrup761** decapsulation core, which it completes in 2471 cycles against Peng’s 3829, a 35.5% core reduction (12.4% once the fixed hash is included). We also compare against a 3-way Karatsuba baseline (Bernstein’s B1), noting that no optimized hardware B1 exists and we do not build one; the comparison there is at the operation-count level only. Synthesizing U1 and B1 as combinational circuits isolates where the five-way advantage comes from. It is not gate count: each $\mathbb{F}_9$ product expands into four $\mathbb{F}_3$ products, so at the same architecture the two use nearly the same area. The gain is in depth, as those four $\mathbb{F}_3$ products run in parallel: 138.27 ps against 57.95 ps for one $\mathbb{F}_3$ multiplier, $2.4\times$ rather than $4\times$. The advantage is thus a hardware effect: it grows with design parallelism and matches the scalar-C software figure across two parameter sets. All Verilog, testbenches, and synthesis scripts are openly available.¹

Keywords: NTRU Prime · post-quantum cryptography · ASIC · hardware implementation · polynomial multiplication · characteristic-three arithmetic · five-way split · Karatsuba · key encapsulation mechanism · sntrup761

1 Introduction

Large-scale quantum computers would undermine the public-key cryptography that today protects internet traffic. Shor’s algorithm, run on such a machine, can solve the integer-factoring and discrete-log problems behind RSA and elliptic curves in polynomial time [20]. This drove

*Independent Researcher, Türkiye (previously Assistant Professor in the Cyber Security department at Copenhagen School of Business and Technology (EK), Denmark). esramath@gmail.com

¹Project repository: <https://github.com/schrodinket/YC-5way-asic-sntrup761>

a multi-year standardization effort at NIST, whose first three outputs appeared in mid-2024: FIPS 203 (ML-KEM), FIPS 204 (ML-DSA), and FIPS 205 (SLH-DSA) [15].

Streamlined NTRU Prime [4] reached the third NIST round as a conservatively designed alternate. Its arithmetic uses the ring $\mathbb{Z}_q[x]/(x^p - x - 1)$ for prime p, q ; being non-cyclotomic, it avoids a structure that certain attacks on other lattice schemes depend on. NIST did not pick it for standardization, though not for any weakness in its security. It reflected a smaller implementation ecosystem and, at the time, the absence of a complete quantum-random-oracle-model (QROM) security proof. That gap has since been closed [3].

NTRU Prime is still in active use. OpenSSH 9.0 (April 2022) made it the default post-quantum key exchange through the hybrid `sntrup761x25519-sha512` suite [17]. TinySSH ships it as well, and GitHub turned it on for SSH access in September 2025 [6]. Although recent OpenSSH releases add an ML-KEM hybrid, the `sntrup761x25519` suite remains enabled and negotiated by default, so the installed base continues to run this decapsulation. Industry measurements report that over 11 million SSH hosts, around 6% of all servers and more than 20% of OpenSSH servers, now support PQC, with NTRU Prime (SNTRUP) usage growing 21% between October 2024 and March 2025 [9]. It reaches beyond SSH: it is standardized in RFC 9941, shipped in Red Hat Enterprise Linux 10, kept as the only NTRU Prime variant in the liboqs library, and used in PQConnect and in commercial VPNs such as Palo Alto’s. It was also proposed as a conservative alternative for TLS 1.3 after the Google–Cloudflare CECQP2 experiment [5]. Its non-cyclotomic ring design makes it a structurally independent hedge alongside cyclotomic schemes such as ML-KEM, the kind of backup NIST recommends against a future structural attack. Such a KEM is also a building block in larger protocols, including threshold and multi-party decryption [23], where faster decapsulation is equally valuable. This deployment, from servers to embedded and IoT endpoints, makes efficient implementation of NTRU Prime on a range of hardware, including dedicated ASIC, worth studying. Hardware acceleration helps on both ends: busy servers that need throughput and small devices that need a compact, fast multiplier.

The characteristic-three multiplication. The decapsulation procedure of Streamlined NTRU Prime performs polynomial multiplications in two distinct rings. The computationally dominant operations are in $\mathcal{R}/q = \mathbb{Z}_q[x]/(x^p - x - 1)$, but decapsulation also requires a characteristic-three multiplication in $\mathcal{R}/3 = \mathbb{Z}_3[x]/(x^p - x - 1)$, with coefficients in $\{-1, 0, 1\}$, which accounts for a substantial share of the decapsulation cost. This characteristic-three product is the target of the algorithm we evaluate.

We measure that share directly, in the setting our design targets. Our multiplier is unmasked, so the reference point should be an unmasked decapsulation. We instrumented the open-source decapsulation of Peng et al. [18], added per-state cycle counters, and ran it under a known-answer test for `sntrup761`. The $\mathcal{R}/3$ multiplication accounts for 1528 of 3829 working cycles, that is **39.9%** of the decapsulation core (Figure 1). The full per-state breakdown is in Section 5.

For context, the only prior NTRU Prime ASIC, the masked design of Land et al. [11], reports 31% for the same product. The unmasked share is higher, and this is expected. Masking adds a proportional overhead to every step of the decapsulation, and the $\mathcal{R}/q$ path is relatively cheaper without it, so the characteristic-three product carries more weight in an unmasked design. We use our own 39.9% throughout, since it is the figure that matches what we build.

Efficient $\mathbb{F}_3$ polynomial multiplication has its own line of research. The schoolbook method and the Karatsuba 2-way split are classical baselines. Cenk, Hasan, and Zadeh introduced 3-way split methods over $\mathbb{F}_3$ [7]. Bernstein et al. introduced a 3-way split (B1) exploiting special evaluation points and integrated it into Streamlined NTRU Prime [5]. Yeniaras and Cenk then introduced a 4-way split and an unbalanced 5-way split (U1). Their hybrid combination (U1-hybrid) reduces arithmetic complexity over $\mathbb{F}_3$. It achieves a 29.85% scalar-C speedup for `sntrup653` and a 35.52% scalar-C speedup for `sntrup761` over B1 on x86 [22, 21].

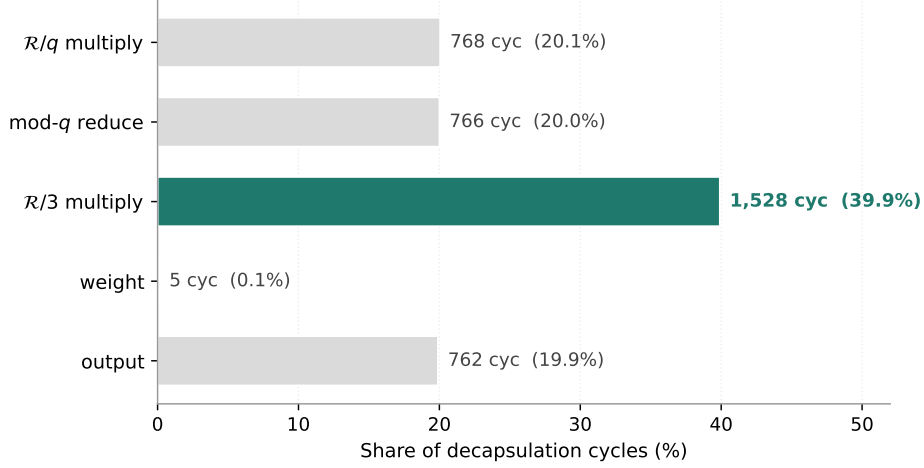


Figure 1: Measured cost breakdown of one *unmasked* `sntrup761` decapsulation core, from cycle-accurate simulation of the open-source design of Peng et al. [18] under a known-answer test (Section 5.6). The single $\mathcal{R}/3$ multiplication, our target, takes 1528 of 3829 cycles, or 39.9%. This is the setting our multiplier targets. The masked ASIC of Land et al. [11] reports 31% for the same product; masking adds a proportional overhead to every step, so the unmasked share is higher. The number-theoretic transform applies to $\mathcal{R}/q$ but not to $\mathcal{R}/3$, so the characteristic-three product relies on schoolbook, Karatsuba, or a split method.

The gap. Every prior NTRU Prime hardware implementation computes the $\mathcal{R}/3$ product with a *schoolbook* or *two-way Karatsuba* multiplier (Section 3). No split beyond two-way has reached a hardware evaluation. Whether the arithmetic advantage of the Yeniaras-Cenk 5-way multiplier carries over to an ASIC, where the cost model differs fundamentally from software, is an open question, and the one this paper answers.

Contributions.

- The first silicon-level study of a higher-radix (5-way) characteristic-three multiplier for NTRU Prime, one that is neither schoolbook nor 2- or 3-way Karatsuba.
- A hierarchical, exact-inventory latency methodology: operation counts taken from instrumented reference C code and cross-validated by recursive static analysis, combined with measured Nangate 45 nm critical-path delays. All Verilog blocks are functionally verified (300/300 random tests each).
- Synthesis of all three degree-761 $\mathcal{R}/3$ multipliers to Nangate 45 nm in deployable form, one per arithmetic class. Area alone ranks them in the wrong order (Peng 0.32, U1 0.43, B1 0.90 mm²); the area-delay product (ADP) does not, and there U1 leads by 6.8 $\times$ over the schoolbook and 5.1 $\times$ over B1.
- An arithmetic isolation: synthesized as fully combinational circuits, U1 and B1 cost nearly the same area (2.71 vs. 2.45 mm²) despite a 3.34 $\times$ gap in multiplication count, because each $\mathbb{F}_9$ product of U1 expands into four $\mathbb{F}_3$ products. The higher-radix advantage is therefore not in gate count but in depth.
- A four-scenario, architecture-matched comparison of U1-hybrid against a 3-way Karatsuba baseline (B1) for `sntrup761`. Since B1 has no optimized hardware implementation in the literature and building one is out of our scope, this comparison is at the operation-count level only; the Yeniaras-Cenk 5-way multiplier has the lower count in every scenario (15.6%

Algorithm 1: The `sntrup761` decapsulation procedure. Line 4 is the $\mathcal{R}/3$ product that this paper accelerates; two $\mathcal{R}/q$ products (lines 2 and 6) surround it.

Input: ciphertext C , secret key $\text{sk} = (f, 1/g, h, \rho)$
Output: session key

```

1  $c \leftarrow \text{Decode}(C)$  // ciphertext into  $\mathcal{R}/q$ 
2  $t \leftarrow c \cdot (3f) \in \mathcal{R}/q$  // first  $\mathcal{R}/q$  product
3  $e \leftarrow (\text{Round}(t) \bmod 3) \in \mathcal{R}/3$  // reduce into  $\mathcal{R}/3$ 
4  $r' \leftarrow e \cdot (1/g) \in \mathcal{R}/3$  // the  $\mathcal{R}/3$  product we target
5  $r' \leftarrow \text{Lift}(r') \in \mathcal{R}/q$ 
6  $c' \leftarrow \text{Round}(h \cdot r') \in \mathcal{R}/q$  // second  $\mathcal{R}/q$  product, for re-encryption
7  $C' \leftarrow (\text{Encode}(c'), \text{HashConfirm}(r', h))$ 
8 if  $C' = C$  then
9 |   return  $\text{HashSession}(1, r', C)$  // ciphertext accepted
10 else
11 |   return  $\text{HashSession}(0, \rho, C)$  // reject, use  $\rho$ 
12 end
```

to 64.6%). Our main comparison in this work is with the optimized and published parallel schoolbook multiplier of Peng et al., the state of the art for this operation.

- A comparison against the two $\mathcal{R}/3$ arithmetic classes seen in prior NTRU Prime work: schoolbook and Karatsuba. Prior hardware uses schoolbook (Peng [18], Land [11], Marotzke’s thesis [13]) or a two-way Karatsuba (Marotzke’s FPGA design [12]); the 3-way Karatsuba appears only in software (Bernstein’s B1 [5], pqm4 [1]) and serves as our baseline here. We also clarify that the pqm4 routine labelled “Toom-Cook” [1] is, at the assembly level, a 3-way Karatsuba.
- Confirmation that the scalar-C software speedup (35.52%) reproduces on hardware (36.9% pure-multiplication), so the arithmetic advantage carries from software to silicon.
- Open, reproducible artifacts: all sources, synthesis scripts, and the latency computation are available in the repository.¹

2 Background

2.1 Streamlined NTRU Prime and its decapsulation

The scheme. The scheme is a KEM with parameters p, q (both prime) and a weight bound w . Its arithmetic lives in $\mathcal{R}/q = \mathbb{Z}_q[x]/(x^p - x - 1)$, an irreducible and non-cyclotomic ring. We focus on `sntrup761`, which sets $p = 761$, $q = 4591$, $w = 286$ for NIST Category 2 and is the suite OpenSSH 9.0 defaults to.

The two multiplications. Decapsulation performs (i) a dominant $\mathcal{R}/q$ multiplication, extensively optimized in prior work via the number-theoretic transform (NTT), Toom-Cook, and Karatsuba; and (ii) an $\mathcal{R}/3$ multiplication between a small polynomial and the secret-key inverse $g^{-1} \in \mathcal{R}/3$, with both operands in $\{-1, 0, 1\}$. Our work targets (ii). The NTT does not carry over to $\mathbb{Z}_3$, which lacks the roots of unity it needs, so hardware designs fall back on schoolbook or Karatsuba for this product.

Characteristic-three arithmetic. We work over $\mathbb{F}_3 = \{-1, 0, 1\}$ and its degree-two extension $\mathbb{F}_9 = \mathbb{F}_3[\xi]/(\xi^2 + 1)$. A sub-product is an $\mathbb{F}_3$ product when both operands lie in $\mathbb{F}_3$, and an

$\mathbb{F}_9$ product otherwise. The cost difference between the two is what drives the analysis: one $\mathbb{F}_9$ multiplication expands into four $\mathbb{F}_3$ multiplications and two $\mathbb{F}_3$ additions. This expansion is the crux of the paper. In serial or single-instruction multiple-data (SIMD) software it is a cost, but in parallel hardware, as we show, it is almost free.

2.2 Polynomial multiplication methods

The hybrid builds the $\mathcal{R}/3$ product from three methods, applied recursively down to a schoolbook base. We define them here so the paper is self-contained; the full derivations are in the original work [22, 21]. Each multiplies two degree- $(n-1)$ polynomials, and we write $M_3(n)$ and $M_9(n)$ for the number of $\mathbb{F}_3$ and $\mathbb{F}_9$ operations, with subscripts $\otimes$ and $\oplus$ for multiplications and additions.

Schoolbook (SB). The direct method multiplies every pair of input coefficients, so $M_{3,\otimes}(n) = n^2$ over $\mathbb{F}_3$ and $M_{9,\otimes}(n) = n^2$ over $\mathbb{F}_9$. It is the base case of the recursion, used once the blocks are small enough that splitting no longer pays ($n \leq 16$ here).

Two-way Karatsuba (KA2, UB). Each operand is split in two, $A = A_0 + yA_1$, and the product is formed from three half-size multiplications $P_0 = A_0B_0$, $P_1 = (A_0+A_1)(B_0+B_1)$, $P_2 = A_1B_1$, trading one multiplication for additions. The recursion and closed form over $\mathbb{F}_3$ are

$$M_3(n) = 3M_3(n/2) + \frac{7n}{2} - 3, \quad M_3(n) = 6.5 n^{\log_2 3} - 7n + 1.5,$$

with $M_{3,\otimes}(n) = n^{\log_2 3}$ and $M_{3,\oplus}(n) = 5.5 n^{\log_2 3} - 7n + 1.5$; the same recursion over $\mathbb{F}_9$ gives $M_9(n)$ with the $\mathbb{F}_9$ operation counts. This is the improved balanced form KA2 of Bernstein [21], about 7% cheaper than plain Karatsuba ($7 n^{\log_2 3} - 8n + 2$), and it applies when n is even. The unbalanced variant UB [21] handles any n by splitting into parts of size $(n+1)/2$ and $(n-1)/2$,

$$M_3(n) = 2M_3\left(\frac{n+1}{2}\right) + M_3\left(\frac{n-1}{2}\right) + \frac{7(n-1)}{2},$$

where one product is free because the two top coefficients coincide. UB reduces to KA2 when n is even, so the two are one 2-way method with UB the general case. UB meets the block sizes the 5-way split leaves behind (31 and 29 in the `sntrup761` tree).

Yeniaras-Cenk 5-way split (V1, U1). The product is evaluated at nine points of $\mathbb{F}_9$, $\{0, 1, \xi, -\xi, \xi+1, -\xi+1, -\xi-1, \xi-1, \infty\}$, giving nine sub-products $H_0, \dots, H_8$. The points come in conjugate pairs over $\mathbb{F}_9$, so three of the nine are redundant, leaving six distinct multiplications. V1 [21] splits each operand into five equal blocks. U1 [22] is the unbalanced version: it lets the five blocks differ in size through a parameter k , with recursions over $\mathbb{F}_9$ and $\mathbb{F}_3$

$$\begin{aligned} M_9(n) &= 8M_9\left(\frac{n+k}{5}\right) + M_9\left(\frac{n-4k}{5}\right) + \frac{196n}{5} + \frac{76k}{5} - 72, & M_9(1) &= 6, \\ M_3(n) &= 2M_3\left(\frac{n+k}{5}\right) + M_3\left(\frac{n-4k}{5}\right) + 3M_9\left(\frac{n+k}{5}\right) + \frac{72n}{5} + \frac{42k}{5} - 29, & M_3(1) &= 1, \end{aligned}$$

the $\mathbb{F}_3$ recursion carrying an M_9 term because the evaluation is over $\mathbb{F}_9$. U1 reduces to V1 when n is a multiple of five (the $k = 5$ setting), so V1 is the equal-split special case and the two share the same asymptotic order,

$$M_9(n) = 47 n^{\log_5 9} - 49n - 9, \quad M_3(n) = 23.5 n^{\log_5 9} - 13 n^{\log_5 3} - 37.5n + 28,$$

with $\log_5 9 \approx 1.365$, below the $\log_2 3 \approx 1.585$ of Karatsuba. U1 thus has the lowest arithmetic cost of the methods here and the fastest measured software run-time, including against Bernstein's 3-way B1.

Yeniaras-Cenk hybrid 5-way multiplier (U1-hybrid). U1-hybrid is not a single method but a recursive schedule over the three methods above. At each step it splits the current block with whichever method has the lowest operation count for that size, so the multiplier is a tree whose branches call different methods (Figure 2). For **sntrup761** the tree is the 5-way U1 split at the top two levels (761 and 153), two-way Karatsuba for the intermediate blocks (31 and 29), and schoolbooks at $n \leq 16$. The block sizes depend on the starting degree p , so the tree differs across parameter sets: **sntrup653** instead passes through a 3-way step [22] and is more addition-heavy, a difference we return to in Section 5.

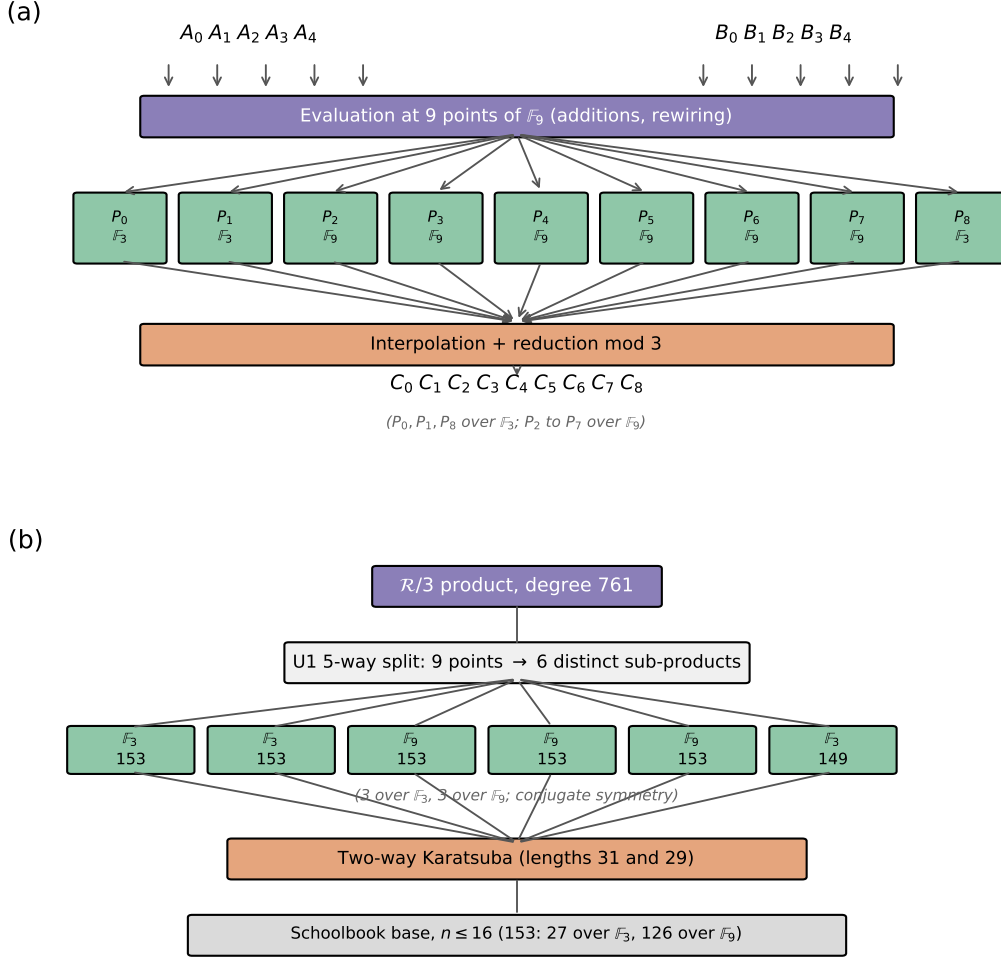


Figure 2: Structure of the Yeniaras-Cenk hybrid 5-way multiplier. (a) One level: each operand is split into five blocks and the product is evaluated at nine points, giving nine sub-products $P_0, \dots, P_8$. Three of them (P_0, P_1, P_8) are plain $\mathbb{F}_3$ products; the other six (P_2-P_7) are $\mathbb{F}_9$ products. The results are then interpolated and reduced modulo 3. (b) In hardware, conjugate symmetry pairs the six $\mathbb{F}_9$ products, so only three of them are distinct: the level therefore has three $\mathbb{F}_3$ and three $\mathbb{F}_9$ sub-multiplications, of length 153 (and one of length 149). Each recurses through two-way Karatsuba (block lengths 31 and 29) to a schoolbook base case at $n \leq 16$, giving 153 schoolbook calls (27 over $\mathbb{F}_3$, 126 over $\mathbb{F}_9$), the inventory used in our latency model (Section 4). The **sntrup653** tree has the same 5-way split but reaches its base cases through a 3-way step [22], ending in $n = 9$ schoolbooks.

3 Related Work: NTRU Prime Implementations and $\mathcal{R}/3$ Multiplication

We survey NTRU Prime hardware and embedded implementations, identifying which $\mathcal{R}/3$ multiplication strategy each uses (Table 1), and then position our contribution. Throughout, we distinguish the $\mathcal{R}/q$ multiplication (mod $q = 4591$, the dominant cost, amenable to NTT/Toom–Cook) from the $\mathcal{R}/3$ multiplication (mod 3, our target, for which NTT does not directly apply).

Embedded software. Several works target embedded software. Cheng et al. [8] gave an early AVR (ATmega1284) implementation of `sntrup653`, using recursive Karatsuba in hand-written assembly for the polynomial product. On the ARM Cortex-M4, the pqm4 framework [10] set up systematic benchmarking; its $\mathcal{R}/3$ routine is a 3-way Karatsuba (clarified below). Alkim et al. [1] sped up the $\mathcal{R}/q$ product with NTT-based methods (Good’s trick and mixed-radix NTTs, giving 17–32% faster multiplication), and Alkim, Hwang, and Yang [2] extended this to more parameter sets. None of these optimized the $\mathcal{R}/3$ product itself: they reused whatever the existing implementation provided.

x86/ARM vectorized software. Bernstein et al. [5] (OpenSSLNTRU, AVX2) introduced the 3-way split (B1) for $\mathbb{F}_3$ and integrated it into TLS 1.3. A line of work by Hwang and co-authors pushed vectorized polynomial multiplication for NTRU Prime further on Intel/AMD and 64-bit ARM. These target the $\mathcal{R}/q$ product or the vector-friendly regime; B1 is the relevant $\mathcal{R}/3$ baseline and the direct predecessor of the algorithm we evaluate. B1 has no published hardware implementation: it was introduced as AVX2 software. The B1 hardware we synthesize here is therefore new, built only to place both algorithms on the same architecture for an arithmetic comparison. A fully optimized, pipelined B1 is a separate project and out of scope for this U1-focused work. Our U1-vs-B1 comparison rests on operation count and the four-scenario model, which hold the architecture fixed and do not depend on any pipelining of B1.

FPGA. Marotzke [12] (CARDIS 2020) gave the first full constant-time FPGA implementation, using a combination of Karatsuba and schoolbook multiplication for $\mathcal{R}/3$, optimized for low area (1841 slices on a Zynq Ultrascale+). Peng et al. [18] (JCEn 2023) reported the fastest FPGA design so far: a decapsulation of 10,989 cycles running at 285 MHz. Their $\mathcal{R}/3$ product uses the same module (`rq_mult_parallel`) as the $\mathcal{R}/q$ products. It is a schoolbook multiply-accumulate over one fused cell, streaming an operand from block RAM and iterating $2p$ times: one pass to load, one to accumulate. The name is a little misleading. The unit is parallel in the sense that it feeds a wide accumulator, but the p^2 coefficient products are issued sequentially, which is why the $\mathcal{R}/3$ step costs 1528 cycles. Their NTT-plus-CRT (Chinese remainder theorem) multiplier is kept for the $\mathcal{R}/q \cdot \mathcal{R}/q$ products, which show up only in key-generation batch inversion and never in decapsulation. We re-synthesize `rq_mult_parallel` in Section 5. Neisarian and Kavun [16] (2025) proposed MT-TMVP, a modular tiled Toeplitz matrix–vector product multiplier optimizing the area–delay product on FPGA. Marotzke’s PhD thesis [13] (2025) consolidates and extends this line, reporting an even faster design (encapsulation in 2571 cycles at 400 MHz). All of these compute $\mathcal{R}/3$ with schoolbook, Karatsuba, or TMVP.

ASIC. The ASIC literature on NTRU Prime is thin. To our knowledge Land et al. [11] (TCHES 2024) is the *only* prior NTRU Prime ASIC. They give an arbitrary-order, gate-level masked `sntrup761` decapsulation, built for both an Artix-7 FPGA and a 45 nm ASIC, with formally proven side-channel security. Their design starts from the Peng et al. codebase and masks it gate by gate.

Their $\mathcal{R}/3$ multiplier is a schoolbook as well, and the way they build it is worth stating precisely, because it is the closest thing in the literature to a characteristic-three multiplier

designed for silicon. They exploit the fact that a product of two signed two-bit coefficients has only nine input combinations and three output values, and give it a direct Boolean form: with $e = e[0] - 2e[1]$ and $v = v[0] - 2v[1]$, the product $r = r[0] - 2r[1]$ is $r[0] = e[0] \wedge v[0]$ and $r[1] = e[0] \wedge v[0] \wedge (e[1] \oplus v[1])$. Each coefficient multiplication is therefore two AND gates, and in the masked design each of those becomes an HPC2 gadget that consumes fresh randomness. The structure is a plain schoolbook: p^2 such products, no split, no recursion. They report that the two $\mathcal{R}/q$ multiplications consume 62% and the single $\mathcal{R}/3$ multiplication 31% of the decapsulation cycles.

This has a direct bearing on our result. In a gate-level masked design, a coefficient multiplication costs not two AND gates but two *gadgets*, each with its own randomness and latency. The number of coefficient multiplications is therefore what drives area and randomness demand. U1-hybrid performs $3.34\times$ fewer of them than the 3-way Karatsuba and 94.6% fewer than schoolbook. That breakdown is useful to us for two reasons. It confirms that $\mathcal{R}/3$ is a substantial and separately identifiable cost even in a masked design, and it fixes the technology against which an ASIC result for this scheme is naturally read. We synthesize to the same Nangate 45 nm library [14] for that reason.

The thinness of the ASIC line is itself informative. All the optimization effort on the $\mathcal{R}/3$ product has gone into FPGA and into embedded software, where the cost model rewards a high clock or a small instruction count. An ASIC charges for critical-path delay and for standard-cell area, and it charges for every coefficient multiplication in silicon. That is precisely the regime in which an operation-count reduction should pay, and precisely the regime that has not been examined for this product.

Clarification on the pqm4 “Toom-Cook”. The pqm4 routine commonly labelled “Toom-Cook” computes the $\mathcal{R}/3$ product with a 3-way *Karatsuba* at the assembly level. Its top-level routine issues exactly six size-256 sub-product calls, whereas a Toom-3 would issue five. We confirm this in the archived source (`data/pqm4_r3_asm/polymul_768x768_mod3.S`), where the six `b1 gf_polymul_256x256_mod3` calls are the six sub-products of a 3-way Karatsuba.¹ Bernstein’s B1 is likewise a 3-way Karatsuba over the same $761 \rightarrow 768$ padding. A single 3-way Karatsuba baseline, represented here by our synthesized B1 module, therefore stands in for this entire family. The pqm4 form unrolls a single Karatsuba layer over a 256-point Comba base for the M4, which is not area-appropriate for ASIC; its ASIC-appropriate recursive form is B1.

Our contribution against this landscape. Across every NTRU Prime hardware implementation, both FPGA and the single prior ASIC, the $\mathcal{R}/3$ multiplication uses schoolbook, Karatsuba, or TMVP. To the best of our knowledge, no published NTRU Prime hardware implementation has evaluated a higher-radix (4-way or 5-way) characteristic-three split, despite such methods being the state of the art in software. We fill this gap with the first ASIC evaluation of the Yeniaras-Cenk 5-way multiplier. Our main comparison is against the state-of-the-art published multiplier, the parallel schoolbook of Peng et al., re-synthesized on our flow. We also report a 3-way Karatsuba baseline (B1); since no optimized hardware B1 exists, that comparison is at the operation-count level and is not a wall-clock claim.

Arithmetic classification of the prior strategies. Although the implementations above differ in their hardware dataflow, they reduce to only two distinct *arithmetic* classes for $\mathcal{R}/3$. (i) *Schoolbook*, covering three designs: Peng et al.’s parallel multiply-accumulate (MAC) multiplier, Land et al.’s masked schoolbook, and the Toeplitz matrix-vector product (TMVP) of Neisarian and Kavun. The last warrants a note. A TMVP is the matrix form of schoolbook multiplication and performs the *same* number of coefficient multiplications and additions. The contribution of MT-TMVP is FPGA-specific resource optimization, namely BRAM tiling and area-delay product, not a reduction in arithmetic complexity. On an ASIC it therefore offers no

Table 1: NTRU Prime hardware and embedded implementations, classified by the *arithmetic* method each uses for the characteristic-three ($\mathcal{R}/3$) multiplication. Despite differing hardware dataflows, all prior work reduces to two arithmetic classes: schoolbook (S) and Karatsuba (K). TMVP is the matrix form of schoolbook and has identical operation count (class S). Our work is the first to use a higher-radix split (5-way).

Work	Year	Platform	$\mathcal{R}/3$ method	Class
Cheng et al. [8]	2020	AVR ATmega	recursive Karatsuba	K
Marotzke [12]	2021	FPGA	Karatsuba + schoolbook	K
Alkim et al. [1]	2021	Cortex-M4	3-way Karatsuba [†]	K
Bernstein B1 [5]	2022	AVX2	3-way split	K
Peng et al. [18]	2023	FPGA	parallel schoolbook	S
Land et al. [11]	2023	FPGA+ASIC 45nm	schoolbook (masked)	S
Neisarian-Kavun [16]	2025	FPGA	TMVP (Toeplitz) [‡]	S
Marotzke [13]	2025	FPGA	schoolbook	S
This work	2026	ASIC 45nm	5-way + Karatsuba	5-way

Class S = schoolbook, K = Karatsuba. [†] Labelled “Toom-Cook” in pqm4; the assembly is a 3-way Karatsuba (six size-256 sub-products, confirmed in our artifacts). [‡] TMVP is the matrix form of schoolbook: identical operation count, with FPGA-specific resource optimization only.

operation-count advantage over schoolbook. (ii) *Karatsuba*: the Karatsuba-plus-schoolbook of Marotzke, the pqm4 3-way Karatsuba, and Bernstein’s B1. These are the only two arithmetic classes in the prior work: schoolbook, used in every prior hardware design, and 3-way Karatsuba, used in software. Our two baselines, naive schoolbook and 3-way Karatsuba (B1), therefore cover the entire prior design space for $\mathcal{R}/3$ at the algorithmic level. No further multiplier needs separate ASIC modelling to complete the comparison.

Scope: ASIC versus FPGA. We deliberately restrict this work to ASIC. ASIC and FPGA use incommensurable cost models. ASIC is measured in picosecond critical-path latency and standard-cell area. FPGA is measured in clock-cycle counts at a synthesis-dependent $f_{\max}$ and in LUT/DSP/BRAM utilization. We therefore do not compare our ASIC latency against the cycle counts of the FPGA designs above, and we keep the latency comparisons throughout platform-consistent.

4 Methodology

4.1 Synthesis of the complete multiplier

We synthesize the complete, connected degree-761 Yeniaras-Cenk 5-way multiplier (`u1_r3_761`) to the target library. The design maps to 2,437,492 cells and a cell area of 2.71 mm^2 , which is 3.39 million gate equivalents at the Nangate 45 nm NAND2 reference of $0.798 \mu\text{m}^2$ (Section 5.1).

Two remarks on how this synthesis is run. First, we synthesize hierarchically rather than on a flattened netlist. A flattened run on the complete multiplier exhausts the memory of our machine during the `flatten` pass and does not reach technology mapping, which is a consequence of the size of the connected tree. Hierarchical synthesis keeps the module boundaries, so the mapper cannot optimize across them, and one may ask how much that costs. We measured it on a representative intermediate block. Flattening `ub_complex_31` yields $58,929 \mu\text{m}^2$ against $58,709 \mu\text{m}^2$ hierarchically, a difference of 0.37%, and in the *opposite* direction to what one would expect. Cross-module optimization finds essentially nothing here. The blocks are already dense internally and they share no common sub-expressions, since each one consumes different coefficients. The hierarchical area is therefore not a loose upper bound. It is the number.

Second, the latency model of the next sections is built on measured per-block costs rather than on this single number. Each distinct block is synthesized once, its delay and area are measured, and these measured costs are combined with the exact call counts of Section 4.2. Every quantity entering the model is measured; the model only supplies the arithmetic that adds them up. This is the same module-by-module reporting that the prior ASIC work on this scheme uses [11], and it is what lets us separate the multiplication layer from the recombination layer. The full synthesis above is the check that the blocks do compose into a real circuit.

4.2 Exact operation inventory

We extract operation counts directly from the reference C code (`data/hyb_2_u1.c`, `data/hyb_2_b1.c`, instrumented by `scripts/instrument.py`; all such files and scripts are in the repository¹) by injecting counters and running the program, and we cross-validate by recursive static analysis. For `sntrup761`, U1-hybrid issues 153 schoolbook calls, 27 over $\mathbb{F}_3$ and 126 over $\mathbb{F}_9$, and B1 issues 405 $\mathbb{F}_3$ schoolbook calls. The corresponding $\mathbb{F}_3$ -multiplication counts are 31,256 (U1, counting each $\mathbb{F}_9$ product as one parallel unit) and 104,472 (B1).

4.3 Measured delays

Each Verilog block is synthesized to the Nangate 45 nm open-cell library [14] with Yosys 0.33 [19], and the critical-path delay is obtained from ABC `stime` on the flattened netlist (the `measure_*` scripts in `scripts/`¹). Table 2 lists the measured delays. The key measurement is `f9_mul` (Figure 3): at 138.27 ps it is about one multiply plus one add deep, confirming the four $\mathbb{F}_3$ multiplies execute in parallel (a serial layout would need $\gtrsim 232$ ps).

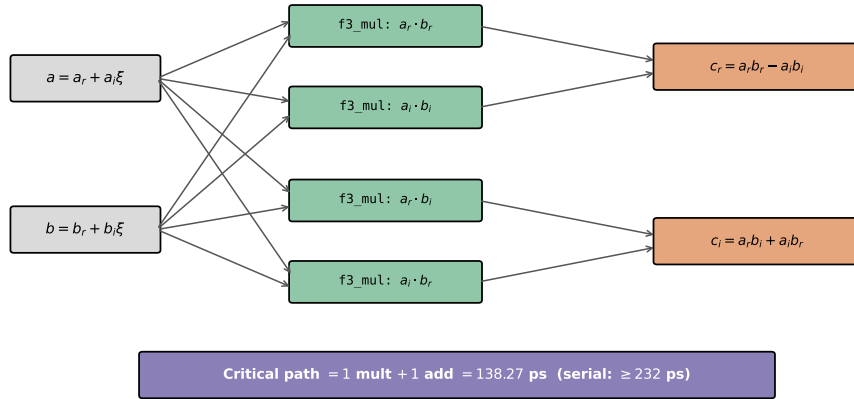


Figure 3: Parallel hardware layout of one $\mathbb{F}_9$ multiplication. The four $\mathbb{F}_3$ multiplications execute in one combinational layer (57.95 ps each); one $\mathbb{F}_3$ addition layer (89.83 ps) then combines them. The critical path is one multiply plus one add (138.27 ps measured), not four multiplies in series ($\gtrsim 232$ ps). Source: `src/vlog/f9_mul.v`.¹

4.4 Synthesized module areas (full tree)

Beyond the primitive cells of Table 2, we synthesized every distinct block of the complete degree-761 U1-hybrid multiplier to the same Nangate 45 nm library with Yosys and ABC, each block flattened internally. All blocks are functionally verified against an independent C/Python golden reference. Table 3 reports the measured cell areas. Two observations stand out. First, an $\mathbb{F}_3$

Table 2: Measured critical-path delays and areas (Nangate 45 nm, flattened, ABC `stime`). All modules functionally verified at 300/300 random tests.[‡]

Cell / block	Delay (ps)	Area (μm^2)
f3_mul (one $\mathbb{F}_3$ mult.)	57.95	21
f3_add (one $\mathbb{F}_3$ add.)	89.83	19
f9_mul (one $\mathbb{F}_9$ mult.)	138.27	149
sbr16_f3 ($\mathbb{F}_3$ schoolbook, $n=16$)	2740.83	7689
sbr17_f3 ($\mathbb{F}_3$ schoolbook, $n=17$)	2929.52	8712
sb16_f9 ($\mathbb{F}_9$ schoolbook, $n=16$)	3104.92	29930

[‡] Verilog and testbenches: `src/vlog/`, `src/testbench/`; `f9_mul` parallel layout in `src/vlog/f9_mul.v`.¹

Table 3: Synthesized cell areas of every distinct block in the full degree-761 U1-hybrid multiplier (Nangate 45 nm, Yosys+ABC, combinational/block-parallel S3 layout). All blocks functionally verified. Source areas: `reports/synth_mult_u1.log`.¹

Layer	Block	Area (μm^2)
Leaf	sb6_f9 ($\mathbb{F}_9$ schoolbook, $n=6$)	2 614
Leaf	sb14_f9 ($\mathbb{F}_9$ schoolbook, $n=14$)	14 752
Leaf	sb15_f9 ($\mathbb{F}_9$ schoolbook, $n=15$)	17 059
Leaf	sb16_f9 ($\mathbb{F}_9$ schoolbook, $n=16$)	19 580
Leaf	sbr14_f3 ($\mathbb{F}_3$ schoolbook, $n=14$)	3 822
Leaf	sbr15_f3 ($\mathbb{F}_3$ schoolbook, $n=15$)	4 395
Leaf	sbr16_f3 ($\mathbb{F}_3$ schoolbook, $n=16$)	5 062
Inter.	ub_complex_31 (2-way Karatsuba $\mathbb{F}_9$)	58 929
Inter.	ub_real_31 (2-way Karatsuba $\mathbb{F}_3$)	15 927
Inter.	ub_complex_29 ($\mathbb{F}_9$)	52 156
Inter.	ub_real_29 ($\mathbb{F}_3$)	13 866
Inter.	ka2_real (balanced Karatsuba $\mathbb{F}_3$)	14 707
Inter.	v1_complex (5-way V1 split $\mathbb{F}_9$)	31 293

schoolbook block costs about a quarter of its $\mathbb{F}_9$ counterpart: **sbr16_f3** takes $5062 \mu m^2$ against **sb16_f9** at $19580 \mu m^2$, a $3.9\times$ ratio. This is expected, since one $\mathbb{F}_9$ product expands into four $\mathbb{F}_3$ products. It also shows why the split’s structure matters in silicon: of the nine sub-products, three are cheap $\mathbb{F}_3$ blocks and only the remaining six are the expensive $\mathbb{F}_9$ blocks, which conjugate symmetry then reduces to three distinct ones. Second, the per-block areas grow with split arity, as expected. The Yeniaras-Cenk 5-way V1 split (**v1_complex**) and the two-way Karatsuba blocks (**ub_complex_31**, **ub_complex_29**) dominate the intermediate layer.

4.5 Latency model and design scenarios

Latency is (cycle count) $\times$ (cycle time). The four scenarios cover the design space, and within each one both algorithms run on the *same* architecture. S1 is fully serial, with one shared multiplier and one shared adder, and is the lower bound. S2 parallelizes the recombination additions with logarithmic-depth trees. S4 isolates the multiplication core alone, the minimal architectural assumption. S3 is block-parallel, with every schoolbook a single-cycle combinational block, and is the upper bound. The numbers in this table are produced by `scripts/compute_latency.py` in the repository.¹

5 Results

5.1 Complete $\mathcal{R}/3$ multiplier with reduction

We assembled the full degree-761 Yeniaras-Cenk 5-way multiplier (leaf to root: `ue_v14_real` $\rightarrow$ two `ue_v12_real`, three `ue_v12_complex`, one `ue_v11_real`) and appended polynomial reduction modulo $x^{761} - x - 1$ to obtain a complete $\mathcal{R}/3$ unit (`u1_r3_761`). The reduction admits a single combinational layer: substituting $x^{761} = x + 1$ folds coefficient $i \geq 761$ to positions $i - 761$ and $i - 760$, both of which are below 761 for all $i \leq 1520$ (the maximum folded index is 760). Hence there is no reduction chain, and each output coefficient is a sum of at most three raw coefficients, $\text{out}[k] = \text{raw}[k] + \text{raw}[k+760] + \text{raw}[k+761]$ (modulo 3, with range guards). This contrasts with the serial $\sim p$ -cycle reduction used by prior streaming multipliers. The complete unit passed functional verification against an independent reference (generator `scripts/gen_u1_r3_761.py`, module `u1_r3_761_0603.v`).¹

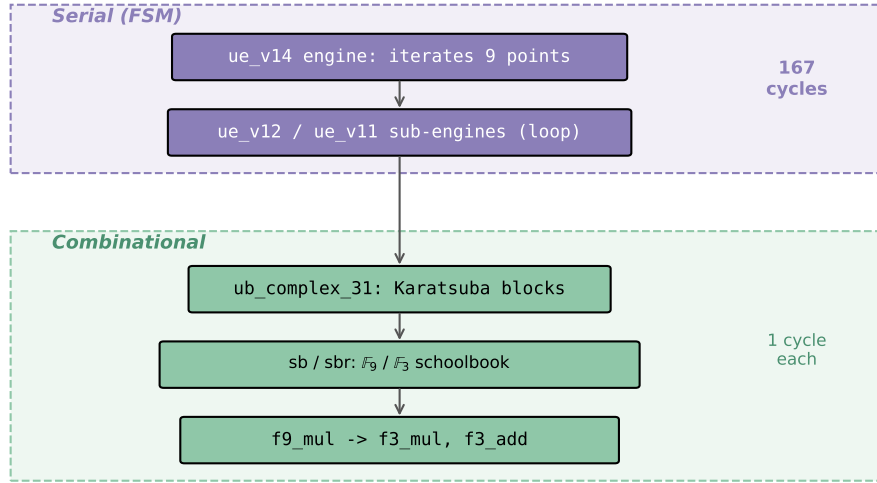


Figure 4: Serial-combinational architecture of the single-engine U1 multiplier. The upper recursion levels are a finite-state machine (FSM) that iterates one shared engine over 167 cycles; the lower levels are combinational, each computing in one cycle. This balances cycle count against area.

Synthesis of the deployable designs. Every area we report is the `chip area` line of a Yosys run; the flow, the log file behind each figure, and the reproduction steps are in Appendix A. We synthesized all three multipliers in the form a designer would actually integrate. For U1 this is the single-engine variant (Figure 4): the upper levels of the recursion are a finite-state machine that iterates one shared engine, and the lower levels are combinational. It completes the $\mathcal{R}/3$ product in 167 cycles. For B1 the same reasoning gives a single serial schoolbook engine cycled 405 times. For the schoolbook class we re-synthesize the published multiplier of Peng et al. [18] (`rq_mult_parallel`), the fastest prior $\mathcal{R}/3$ multiplier, to the same library.² It takes 1528 cycles, one load pass and one multiply-accumulate pass over $p = 761$. Table 4 reports the three.

The ordering is not the one the operation counts predict, and this is the useful part. Peng is the *smallest* design at 0.32 mm^2 , smaller than the Yeniaras-Cenk 5-way multiplier’s 0.43 mm^2

²Their design is mixed VHDL and Verilog. We translate the two Verilog lookup modules to VHDL, elaborate the whole design with GHDL, and synthesize the resulting netlist with the same Yosys and ABC flow used for U1 and B1. Sources are in the repository.¹

Table 4: Synthesis of the complete degree-761 $\mathcal{R}/3$ multipliers to Nangate 45 nm (Yosys 0.33, ABC, hierarchical; flattening changes a representative block by 0.37%, Section 4.1). 1 GE = NAND2_X1 = $0.798 \mu\text{m}^2$. The *deployable* pair is the comparison that matters. The *combinational* pair is not a design; we synthesize it to isolate the arithmetic (see below).

	Deployable			Comb. (arithmetic only)	
	Peng	B1	U1	U1	B1
$\mathcal{R}/3$ method	schoolbook	Karatsuba	YC 5-way	YC 5-way	Karatsuba
Architecture	serial FSM	serial FSM	FSM + comb.	fully comb.	fully comb.
Cycles	1528	405	167	1	1
Coefficient mults	579,121	104,472	31,256	31,256	104,472
Standard cells	40,757	692,200	331,387	2,437,492	2,286,648
Cell area (mm^2)	0.32	0.90	0.43	2.71	2.45
Gate equivalents	401 kGE	1.13 MGE	541 kGE	3.39 MGE	3.07 MGE
Area $\times$ cycles (ADP) ($\text{mm}^2 \cdot \text{cycle}$)	489.0	365.5	72.0	—	—

and less than half of B1’s 0.90 mm^2 . It is also by far the slowest, at 1528 cycles against U1’s 167. A serial schoolbook is a single multiply-accumulate cell iterated $2p$ times, so it is the extreme point of the area–time trade-off: minimal hardware, maximal cycles. Area alone therefore gives a misleading ranking: it makes Peng look best, when the ADP shows it is the worst of the three.

The ADP, by contrast, gives the Yeniaras-Cenk 5-way multiplier the lowest value of the three. The ADP is $72.0 \text{ mm}^2 \cdot \text{cycle}$ against 365.5 and $489.0 \text{ mm}^2 \cdot \text{cycle}$ for B1 and Peng, a factor of $5.1\times$ and $6.8\times$. Against B1 the two effects even compound: the Yeniaras-Cenk 5-way multiplier is $2.09\times$ smaller *and* $2.43\times$ faster. Against Peng the Yeniaras-Cenk design trades $1.35\times$ the area for $9.15\times$ fewer cycles. The same ordering holds on FPGA, where the three occupy 27,763, 41,457, and 70,016 LUTs at 1528, 167, and 405 cycles for Peng, U1, and B1 respectively, so this is a property of the algorithms rather than of the target.

Area and cycle count as the reported metrics. We did run a static-timing analysis on the synthesized netlist, and the result is informative. The worst path reports 141 ns, which would be a clock of about 7 MHz. The path is not deep. It is a single gate driving a load capacitance of roughly 2200 units, and that one hop accounts for 130 of the 141 ns. Inserting buffers with the synthesizer’s generic buffering pass grows the area by 51% while improving the slack by only 0.3 ns, a negligible gain that does not justify the area cost.

This is a limitation of the flow, not of the design. An open-source logic-synthesis flow maps a netlist to cells. It does not do timing-driven buffer-tree insertion, fan-out control, or drive-strength selection, and a high-fanout net of this kind is precisely what those steps exist to fix. A production flow would resolve it. Quoting 7 MHz would therefore report a property of the tool rather than of the multiplier, and we do not do it.

Area and cycle count do not have this problem. Both are properties of the algorithm and the architecture, and all designs pass through the identical flow at the same clock ceiling. Because the cycle time is then a shared constant, the area-delay product reduces to area times cycle count, which is what we report as ADP; the constant cancels in every ratio we compare. A timing-optimized ASIC implementation, with proper physical synthesis and the pipelining it would justify, is future work; this paper evaluates the arithmetic, not a tape-out-ready circuit.

Area of the two parallel variants. The right-hand pair of Table 4 is not a design. Neither fully combinational netlist is something one would fabricate: each lays the entire recursion tree

Table 5: Latency of the Yeniaras-Cenk 5-way multiplier vs. a 3-way Karatsuba baseline (B1) for `sntrup761` across four ASIC design scenarios (Nangate 45 nm). Each scenario applies one architecture to both, so the comparison reflects operation count rather than an optimized B1, which does not exist in the literature.

Scenario	U1 (μs)	B1 (μs)	U1 advantage
S1 Serial (lower bound)	9.557	11.323	15.6%
S2 Parallel recombination	3.823	6.056	36.9%
S4 Pure multiplication	3.821	6.054	36.9%
S3 Block-parallel (upper bound)	0.394	1.115	64.6%

out in silicon at once, and the U1 variant alone is 2.71 mm^2 , comparable to a whole decapsulation core. We synthesize them for one reason, to hold the architecture fixed and see what the arithmetic alone costs.

Held at the same architecture, U1 and B1 come out at 2.71 and 2.45 mm^2 , respectively. The Yeniaras-Cenk design comes out 10.6% ($1.11\times$) *larger*, even though its 31,256 coefficient multiplications are 70.1% ($3.34\times$) fewer than B1’s 104,472. A 70.1% reduction in multiplications has not produced any reduction in gates. The reason is the field. U1’s multiplications are $\mathbb{F}_9$ -valued: 25,020 of its 31,256 products are full- $\mathbb{F}_9$, and each expands into four $\mathbb{F}_3$ multiplications in the netlist. Counted in $\mathbb{F}_3$ multipliers the two designs are within 2% of each other, at 106,316 for U1 against 104,472 for B1, and the synthesized areas track that almost exactly. *The higher-radix split does not buy fewer gates.*

What it buys is depth. An $\mathbb{F}_9$ multiplier occupies about seven times the area of an $\mathbb{F}_3$ one (149 against $21 \mu m^2$, Table 2) but only $2.4\times$ the delay (138.27 against 57.95 ps), because its four $\mathbb{F}_3$ products are computed side by side rather than in sequence. The same gates do more work per unit of time. This is what the deployable Yeniaras-Cenk design turns into a real advantage. Each iteration of the U1 engine resolves a larger part of the tree, so the single-engine design needs 167 cycles where B1 needs 405. It does so in less area, not more.

5.2 The four-scenario latency model

Table 5 and Figure 5 report latency in the four scenarios. The Yeniaras-Cenk 5-way multiplier has the lower latency in every scenario, and its advantage grows with parallelism: from 15.6% in the serial case to 64.6% in the block-parallel one. Scenarios S2 and S4 give the same figure. With a parallel addition tree, the recombination latency ($\sim 1.8 \text{ ns}$) is negligible next to the multiplication core, so the total is set by the multiplications. That is the regime where the fewer multiplications of U1 matter most.

5.3 Comparison with literature $\mathcal{R}/3$ strategies

Table 6 places U1-hybrid against the two strategies used by all prior NTRU Prime hardware, on the same flow. The Yeniaras-Cenk 5-way multiplier has a lower operation count than both naive schoolbook (which scales as n^2) and the 3-way Karatsuba family; against the Peng schoolbook this is also a synthesized area-delay advantage.

5.4 Software–hardware agreement

The pure-multiplication advantage of the Yeniaras-Cenk 5-way multiplier on ASIC (36.9%, S4) closely matches the 35.52% measured in scalar-C software on x86 [22]. The software figure is a scalar-C baseline; optimized assembly, where accumulating multiply-add makes additions nearly free, would narrow it. Our result is a hardware one, shown directly by the synthesis, and does

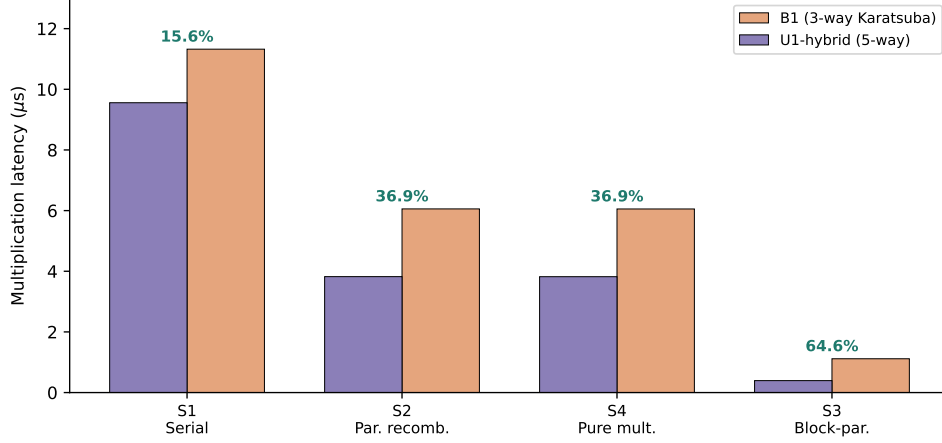


Figure 5: Latency of the Yeniaras-Cenk 5-way multiplier vs. a 3-way Karatsuba baseline (B1) across the four design scenarios; both run on one architecture, so the comparison reflects operation count rather than an optimized B1. The advantage grows with parallelism, from 15.6% in the serial S1 to 64.6% in block-parallel S3. Produced by `scripts/compute_latency.py`.¹

Table 6: U1-hybrid vs. the two arithmetic classes used by all prior NTRU Prime hardware (Nangate 45 nm, $n=761$). Percentages are the operation-count advantage of the Yeniaras-Cenk 5-way multiplier. Class S (schoolbook) covers [18, 11, 13, 16]; class K (Karatsuba) covers [5, 1, 12].

Class	$\mathcal{R}/3$ method	S4 pure mult.	S1 full (serial)
S	Naive schoolbook	$33.56 \mu s$ (+88.6%)	$85.45 \mu s$ (+88.8%)
K	3-way Karatsuba	$6.05 \mu s$ (+36.9%)	$11.32 \mu s$ (+15.6%)
5-way	Yeniaras-Cenk 5-way (U1, this work)	$3.82 \mu s$	$9.56 \mu s$

not rely on the software comparison. Two independent platforms yielding nearly the same figure indicates the advantage is algorithmic and platform-independent: software cycle count scales with operation count, ASIC latency with operation count times per-operation delay, and the U1/B1 ratio is preserved under both.

5.5 Second parameter set: `sntrup653`

We repeated the full procedure for `sntrup653`, using the matching reference C code (the `hyb_1_*.c` files in `data/`¹, with inputs `inp653test/`/`inp654test`) and the same measured cell delays (the Verilog blocks are size-independent). The exact inventory gives U1 20,368 products and 38,808 additions, against B1 77,517 products and 41,112 additions.

As explained in Section 2, the `sntrup653` tree passes through a 3-way step [22] and is therefore more addition-heavy than `sntrup761`. The matching scenario is consequently the full-latency one (S1, which counts both multiplications and additions) rather than the pure-multiplication one. In S1, U1 attains a 28.4% advantage, in close agreement with the 29.85% measured in scalar-C software for `sntrup653` [22]. The agreement holds for both parameter sets, each at the scenario that matches its operation profile (Table 7): `sntrup761` is multiplication-heavy and aligns at S4, `sntrup653` is addition-heavy and aligns at S1. In both cases the Yeniaras-Cenk 5-way multiplier has the lower operation count, and the hardware figure tracks the software figure.

Table 7: U1 advantage in software and ASIC for both parameter sets, each at the scenario matching its operation profile.

Parameter	U1 tree shape	Software (scalar-C)	ASIC
sntrup761	YC 5-way then Karatsuba (mult.-heavy)	35.52%	36.9% (S4)
sntrup653	YC 5-way then 3-way (add.-heavy)	29.85%	28.4% (S1)

5.6 Decapsulation-level measurement

To place the $\mathcal{R}/3$ speedup in the context of a decapsulation, we measured an unmasked design directly rather than relying on the masked profile of Land et al. We instrumented the open-source decapsulation of Peng et al. [18], added per-state cycle counters to its control FSM, and ran a behavioral simulation under a known-answer test for **sntrup761**; the session-key hash matched the reference, confirming functional correctness. Table 8 reports the per-state cycle counts.

Table 8: Measured per-state cycle counts of one unmasked **sntrup761** decapsulation core (Peng et al. [18], behavioral simulation, known-answer test passed).

Decapsulation step	Cycles	Share
$\mathcal{R}/q$ multiplication	768	20.1%
mod- q reduction	766	20.0%
$\mathcal{R}/3$ multiplication	1528	39.9%
weight check	5	0.1%
output	762	19.9%
Total (working states)	3829	100%

In this unmasked design the $\mathcal{R}/3$ multiplication is 39.9% of the decapsulation core, higher than the 31% of the masked ASIC of Land et al. [11]. Masking adds proportional overhead to every step, and the $\mathcal{R}/q$ transform is relatively cheaper here, so the characteristic-three product carries more weight. Because the core is where $\mathcal{R}/3$ dominates, we distinguish two levels below: the decapsulation core (the working states of Table 8), and the complete decapsulation, which adds the fixed-cost hash and encoding. Amdahl’s law makes the two levels concrete. In the core, the $\mathcal{R}/3$ share is 39.9% and the measured $\mathcal{R}/3$ cycle reduction is 89.1% ($1528 \rightarrow 167$), so the core speedup is $0.399 \times 0.891 = 35.5\%$, matching the synthesized cycle count below. The synthesized totals sit a few cycles above the per-state sum of Table 8 (2471 against 2468 for U1, 2720 against 2706 for B1); the difference is the control and bridge handshake that couples each multiplier to the surrounding datapath. At the complete decapsulation level the $\mathcal{R}/3$ share falls to 12.4% of 10,980 cycles once the fixed hash is included, so the same reduction gives a 12.4% complete decapsulation-level speedup. The two figures bound the protocol-level effect.

5.7 Decapsulation synthesis: core and complete

The cycle counts above are one half of the picture. To see the area cost of each choice in a complete design, we synthesized the **sntrup761** decapsulation core for all three multipliers to Nangate 45 nm, using the same flow as before (Appendix A). Each design is the same decapsulation wrapper, with encoding, ciphertext decoding, the control state machine, and the on-chip memories, differing only in the $\mathcal{R}/3$ multiplier. Table 9 reports the result.

Two points on how the areas are counted. U1 and B1 embed their multipliers inside the decapsulation through a bridge, so their wrapper synthesis already includes them. Peng instantiates its multiplier once at a shared top level, outside the decapsulation, so a decap-only

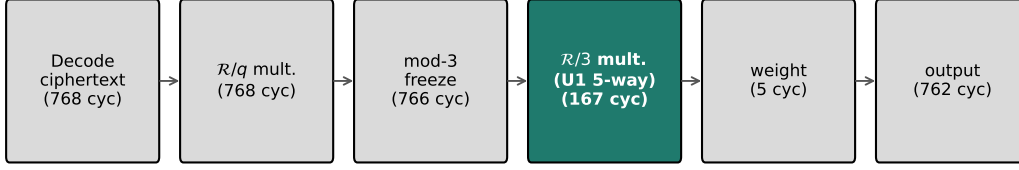


Figure 6: The **sntrup761** decapsulation *core* (the working states, before the fixed SHA-512 hash). All three designs share the same datapath and wrapper and differ in the $\mathcal{R}/3$ multiply, where U1 replaces the schoolbook; the integration of each multiplier adds a few control and bridge cycles, so the totals in Table 9 are slightly above the per-state sum. Per-state cycle counts are for the unmasked Peng et al. design; the highlighted $\mathcal{R}/3$ stage is the one this paper accelerates.

Table 9: **sntrup761** decapsulation core synthesized to Nangate 45 nm (working states of Table 8, before the fixed hash). Each design shares the same encoding, decoding, control, and memory logic and differs only in the $\mathcal{R}/3$ multiplier. ADP is area times cycles. Each *vs. U1* column gives that design relative to U1 on the metric.

Design	Area (mm ²)		Cycles		ADP (mm ² ·cyc)	
	value	vs. U1	value	vs. U1	value	vs. U1
Peng (schoolbook)	0.80	−18%	3829	+55%	3060	+27%
B1 (3-way Karatsuba)	1.44	+48%	2720	+10%	3928	+63%
U1 (Yeniaras-Cenk 5-way)	0.97	—	2471	—	2403	—

synthesis of that design leaves it out. We add its standalone cost (0.32 mm², Section 5) to the 0.48 mm² wrapper. The sum is 0.80 mm², so all three cover the same function.

The Yeniaras-Cenk 5-way multiplier has the lowest ADP of the three. It is 1.27× better than Peng and 1.63× better than B1. The gap is smaller than at the multiplier level. The reason is simple. The encoding, decoding, and control logic is identical in all three and takes a fixed share of the area, so it dilutes any multiplier-level advantage. After this dilution, U1 still completes the core in 2471 cycles against Peng’s 3829, a 35.5% reduction, for only 0.17 mm² more area. The Yeniaras-Cenk 5-way multiplier is faster and only slightly larger, and on the product of the two it has the lowest value. The same ordering holds on FPGA, where the three occupy 29,217, 41,457, and 79,512 LUTs for Peng, U1, and B1 respectively.

5.7.1 Complete decapsulation

The core above excludes the fixed-cost SHA-512 hash and the surrounding encode/decode logic. Adding these gives the complete decapsulation, whose cycle count is dominated by the hash. Since the hash is identical across the three designs, it dilutes the $\mathcal{R}/3$ advantage. The complete decapsulation runs 10,980 cycles for Peng, so the same $\mathcal{R}/3$ change that gives a 35.5% core reduction gives a 12.4% reduction at the complete decapsulation level (Table 10). The two figures bound the protocol-level effect: 35.5% when the surrounding logic is cheap, 12.4% once a full hash is included.

Table 10: Complete `sntrup761` decapsulation cycles (core plus the fixed SHA-512 hash and encode/decode) and the reduction over Peng. The hash is shared across designs, so it dilutes the core-level advantage.

Design	Complete cycles	Reduction vs. Peng
Peng (schoolbook)	10,980	—
B1 (3-way Karatsuba)	9,871	10.1%
U1 (Yeniaras-Cenk 5-way)	9,622	12.4%

6 Discussion

Where the advantage is, and where it is not. The operation counts say that the Yeniaras-Cenk 5-way multiplier performs $3.34\times$ fewer coefficient multiplications than B1. It is tempting to read that as $3.34\times$ fewer gates for the Yeniaras-Cenk design, and the synthesis says otherwise. Laid out at the same architecture, the two designs land within 10.6% of each other. The reason is that U1 counts an $\mathbb{F}_9$ product as one multiplication while the netlist counts it as four $\mathbb{F}_3$ products, and once that is accounted for the two are within 2%: 106,316 against 104,472. The Yeniaras-Cenk higher-radix split over characteristic three does not reduce the amount of $\mathbb{F}_3$ arithmetic. It reorganizes it.

Effect of the reorganization on area and delay. The reorganization helps because the four $\mathbb{F}_3$ products inside an $\mathbb{F}_9$ multiplication are independent. They are computed side by side, not one after another, so an $\mathbb{F}_9$ multiplier costs about seven times the area of an $\mathbb{F}_3$ one but only $2.4\times$ its delay (138.27 against 57.95 ps). The gates are the same; the Yeniaras-Cenk split simply arranges them shallower. This parallelism is what the latency advantages in the synthesized designs come from. In software the same expansion is a penalty. A scalar or SIMD lane runs the four products in sequence and pays for all of them. This is why U1’s software advantage is 35.52% rather than the 70.1% its operation count might suggest.

Impact on the deployable design. The deployable designs make the consequence concrete. A single Yeniaras-Cenk 5-way engine resolves a larger portion of the recursion per iteration than a single B1 engine, because the $\mathbb{F}_9$ block it iterates over does four $\mathbb{F}_3$ products at the depth of one. It therefore needs 167 iterations where B1 needs 405, and the engine that performs them is not larger for it: 0.43 mm^2 against 0.90 mm^2 . The area and the cycle count improve together, by $2.09\times$ and $2.43\times$, and their product by $5.1\times$. This is the deployable-level result for the Yeniaras-Cenk result, and it is stronger than the combinational comparison, which only shows that the arithmetic is a wash.

The four scenarios, read in this light. The scenario model tracks the same mechanism across the design space. The advantage is 15.6% in the fully serial S1, where a single shared multiplier serializes everything and the $\mathbb{F}_9$ parallelism cannot be exploited, and 64.6% in block-parallel S3, where the Yeniaras-Cenk parallelism is fully exploited. S4 at 36.9% is the figure to quote for the Yeniaras-Cenk 5-way multiplier, since it isolates the multiplication core and rests only on measured per-block delays and exact operation counts, and it agrees with the 35.52% measured in scalar-C software. S1 and S3 bound it.

7 Conclusion

We gave the first silicon-level study of a higher-radix characteristic-three multiplier for NTRU Prime, the Yeniaras-Cenk 5-way multiplier. It attains lower `sntrup761` multiplication latency

than the 3-way Karatsuba family in every design scenario (from 15.6% to 64.6%), and at the decapsulation-core level it improves on the parallel schoolbook of Peng et al. in area-delay product (21% lower). The pure-multiplication advantage of the Yeniaras-Cenk 5-way multiplier (36.9%) matches the scalar-C software result. We confirmed the same agreement for `sntrup653` (28.4% ASIC versus 29.85% scalar-C software), where the hybrid recursion takes a different, more addition-heavy shape. All three multipliers synthesize on Nangate 45 nm, so these are real circuits and not only models. In deployable form the schoolbook is the smallest design and the slowest, the Yeniaras-Cenk 5-way multiplier is the fastest, and B1 is neither: 0.32, 0.43, and 0.90 mm² at 1528, 167, and 405 cycles, respectively. Area alone therefore ranks the three arithmetic classes in the wrong order. The ADP, by contrast, gives the Yeniaras-Cenk 5-way multiplier the lowest value, by 6.8× over the schoolbook and 5.1× over B1. The advantage carries into the decapsulation core, though it is smaller there. The shared encoding and control logic takes a fixed share of the area, which dilutes the multiplier-level gap. Even so, U1 has the lowest area-delay product: 1.27× better than schoolbook and 1.63× better than B1. It cuts the core to 2471 cycles from 3829, a 35.5% reduction, for only 0.17 mm² more area. At the complete decapsulation level, once the fixed SHA-512 hash is included, this becomes a 12.4% reduction. Synthesizing U1 and B1 as fully combinational circuits shows why the multiplier-level gap is what it is. At identical architecture they cost nearly the same area, 2.71 against 2.45 mm², even though B1 performs 3.34× more coefficient multiplications, because each $\mathbb{F}_9$ product of U1 expands into four $\mathbb{F}_3$ products in silicon. The Yeniaras-Cenk 5-way split does not save gates. It arranges them so that four multiplications happen at the depth of one, and a single engine then resolves more of the recursion per iteration.

Future work. This work is an arithmetic evaluation of the multiplier. One direction beyond it is masking, which lies outside this scope and is a separate line of work with its own expertise. Because masking cost scales with the number of coefficient multiplications, and the Yeniaras-Cenk 5-way multiplier performs 3.34× fewer of them than B1, a masked U1 could lower both the randomness demand and the area; this matters for deployments where decapsulation runs on untrusted or shared hardware. The masked design of Land et al. [11], on the same open-source Peng codebase [18], would be a natural starting point for whoever undertakes it; gate-level masking is a distinct specialty and we do not attempt it here. The remaining parameter sets, `sntrup857` and above, follow the same hybrid strategy with their own recursion tree.

A Synthesis reproduction

All areas in this paper are the `chip area` lines that Yosys prints after technology mapping. We use Yosys 0.33 with ABC for logic synthesis and mapping, GHDL (`-std=08`) to translate the decapsulation VHDL to Verilog where needed, and the Nangate 45 nm open-cell library (1 GE = `NAND2_X1` = 0.798 μm²). Every design is synthesized hierarchically; a flatten pass on a representative block changes the area by 0.37% (Section 4.1), so the hierarchical figures are not loose bounds.

Table 11 lists, for each synthesized design, the reported cell area and the log file in the artifact repository (<https://github.com/schrodinket/YC-5way-asic-sntrup761>) that contains it. The synthesis scripts that produce these logs, the GHDL translation helpers, and all Verilog and VHDL sources are in the same repository, so the numbers reported here can be reproduced from the sources.

The Peng decapsulation-core figure of 0.80 mm² is the wrapper (479,108) plus the shared multiplier (320,057), since Peng instantiates its multiplier at the top level rather than inside the decapsulation (Section 5.7).

Table 11: Each synthesized design, its cell area (Nangate 45 nm), and the artifact log file with the full report.

Design	Log file (reports/)	Cell area (μm^2)
U1 multiplier (deployable)	<code>synth_mult_u1.log</code>	431,430
B1 multiplier (deployable)	<code>synth_mult_b1.log</code>	902,373
Peng multiplier	<code>synth_mult_peng.log</code>	320,057
U1 combinational	<code>synth_comb_u1.log</code>	2,705,068
B1 combinational	<code>synth_comb_b1.log</code>	2,446,018
U1 decapsulation core	<code>synth_decap_u1.log</code>	972,392
B1 decapsulation core	<code>synth_decap_b1.log</code>	1,443,994
Peng core wrapper	<code>synth_decap_peng.log</code>	479,108

References

- [1] Erdem Alkim, Dean Yun-Li Cheng, Chi-Ming Marvin Chung, Hülya Evkan, Leo Wei-Lun Huang, Vincent Hwang, Ching-Lin Trista Li, Ruben Niederhagen, Cheng-Jhih Shih, Julian Wälde, and Bo-Yin Yang. Polynomial multiplication in NTRU prime: Comparison of optimization strategies on Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):217–238, 2021. doi: 10.46586/tches.v2021.i1.217-238.
- [2] Erdem Alkim, Vincent Hwang, and Bo-Yin Yang. Multi-parameter support with NTTs for NTRU and NTRU prime on Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):349–371, 2022. doi: 10.46586/tches.v2022.i4.349-371.
- [3] Benjamin Benčina, Benjamin Dowling, Varun Maram, and Keita Xagawa. Post-quantum cryptographic analysis of SSH. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 595–613, 2025. doi: 10.1109/SP61157.2025.00126.
- [4] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU prime: Reducing attack surface at low cost. In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography – SAC 2017*, pages 235–260, Cham, 2018. Springer International Publishing. ISBN 978-3-319-72565-9.
- [5] Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, and Nicola Tuveri. OpenSSLNTRU: Faster post-quantum TLS key exchange. In *31st USENIX Security Symposium (USENIX Security ’22)*, pages 845–862. USENIX Association, August 2022. URL <https://www.usenix.org/conference/usenixsecurity22/presentation/bernstein>.
- [6] Brian M. Carlson and Taylor Blau. Post-quantum security for SSH access on GitHub. GitHub Blog, <https://github.blog/engineering/platform-security/post-quantum-security-for-ssh-access-on-github/>, 2025.
- [7] Murat Cenk, Farhad Haghighi Zadeh, and M. Anwar Hasan. New efficient algorithms for multiplication over fields of characteristic three. *Journal of Signal Processing Systems*, 90(3):285–294, 2018. doi: 10.1007/s11265-017-1234-x.
- [8] Hao Cheng, Daniel Dinu, Johann Großschädl, Peter B. Rønne, and Peter Y. A. Ryan. A lightweight implementation of NTRU prime for the post-quantum internet of things. In Maryline Laurent and Thanassis Giannetsos, editors, *Information Security Theory and Practice (WISTP 2019)*, pages 103–119, Cham, 2020. Springer International Publishing. ISBN 978-3-030-41702-4. doi: 10.1007/978-3-030-41702-4_7.
- [9] Forescout Vedere Labs. Q-day countdown: Post-quantum cryptography adoption across devices and industries. Forescout Research Blog, <https://www.forescout.com/blog/q-d>

ay-countdown-new-data-on-post-quantum-cryptography-adoption-across-devices-and-industries/, 2025.

- [10] Matthias J. Kannwischer, Richard Petri, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Post-quantum crypto library for the ARM cortex-m4. <https://github.com/mupq/pqm4>, 2019.
- [11] Georg Land, Adrian Marotzke, Jan Richter-Brockmann, and Tim Güneysu. Gadget-based masking of streamlined NTRU prime decapsulation in hardware. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(1):1–26, Dec. 2023. doi: 10.46586/tches.v2024.i1.1-26. URL <https://tches.iacr.org/index.php/TCHES/article/view/11238>.
- [12] Adrian Marotzke. A constant time full hardware implementation of streamlined NTRU prime. In Pierre-Yvan Liardet and Nele Mentens, editors, *Smart Card Research and Advanced Applications (CARDIS 2020)*, pages 3–17, Cham, 2021. Springer International Publishing. ISBN 978-3-030-68487-7.
- [13] Adrian Marotzke. *Secure and Efficient Hardware Implementations of NTRU Prime*. PhD thesis, Hamburg University of Technology (TUHH), 2025. <https://tore.tuhh.de/entities/publication/e4edad08-8cbd-49a0-b5dd-052105ac0714>.
- [14] NanGate Inc. NanGate FreePDK45 Open Cell Library. <https://si2.org/open-cell-library/>, 2011. 45 nm open standard-cell library.
- [15] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard (fips 203). NIST FIPS 203, 2024. URL <https://doi.org/10.6028/NIST.FIPS.203>. Also FIPS 204 (ML-DSA), FIPS 205 (SLH-DSA).
- [16] Shekoufeh Neisarian and Elif Bilge Kavun. MT-TMVP: Modular tiled TMVP-based polynomial multiplication for post-quantum cryptography on FPGAs. Cryptology ePrint Archive, Paper 2025/1018, 2025. URL <https://eprint.iacr.org/2025/1018>. Also: 25th Int. Conf. on Embedded Computer Systems (SAMOS XXV).
- [17] OpenSSH Project. OpenSSH 9.0 release notes. <https://www.openssh.com/txt/release-9.0>, 2022.
- [18] Bo-Yuan Peng, Adrian Marotzke, Ming-Han Tsai, Bo-Yin Yang, and Ho-Lin Chen. Streamlined NTRU prime on FPGA. *Journal of Cryptographic Engineering*, 13(2):167–186, 2023. ISSN 2190-8516. doi: 10.1007/s13389-022-00303-z. URL <https://doi.org/10.1007/s13389-022-00303-z>. Online first 2022; code at https://github.com/AdrianMarotzke/SNTRUP_on_FPGA.
- [19] Claire Wolf. Yosys Open SYnthesis Suite. <https://yosyshq.net/yosys/>, 2023. Version 0.33.
- [20] Esra Yeniaras. *Quantum resistance*, pages 178–184. Edward Elgar Publishing, Cheltenham, UK, 2026. ISBN 9781035339969. doi: 10.4337/9781035339969.00042. URL <https://www.elgaronline.com/view/book/9781035339969/chapter36.xml>.
- [21] Esra Yeniaras and Murat Cenk. Faster characteristic three polynomial multiplication and its application to NTRU prime decapsulation. *Journal of Cryptographic Engineering*, 12(3):329–348, 2022. ISSN 2190-8516. doi: 10.1007/s13389-021-00282-7. URL <https://doi.org/10.1007/s13389-021-00282-7>. Preliminary version: IACR ePrint 2020/1336.

- [22] Esra Yeniaras and Murat Cenk. Improved polynomial multiplication algorithms over characteristic three fields and applications to NTRU prime. In *Innovative Security Solutions for Information Technology and Communications (SecITC) 2021*, volume 13195 of *LNCS*, pages 125–144. Springer, 2022. ISBN 978-3-031-17510-7. doi: 10.1007/978-3-031-17510-7_9.
- [23] James Hsin yu Chiang, Bernardo David, Tore Kasper Frederiksen, Arup Mondal, and Esra Yeniaras. Detecting rogue decryption in (threshold) encryption via self-incriminating proofs. Cryptology ePrint Archive, Paper 2024/794, 2024. URL <https://eprint.iacr.org/2024/794>.