

PairSwitch: Efficient Galois-Paired Key Switching for Gentry–Lee Matrix FHE

Zhenyu Xiong, Mingsheng Wang, and Han Wang

Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{xiongzhenyu, wangmingsheng, wanghan}@iie.ac.cn

Abstract. Matrix-native fully homomorphic encryption, introduced by Gentry and Lee [10] (Crypto’26), multiplies encrypted matrices with a Trace product and returns its four-component output to an ordinary ciphertext with a BigSwitch, whose keys over an extended ring dominate the key material of the scheme. Existing BigSwitches follow two routes: they either store one key per source, as in the original construction, or factor the secret through the trace and relinearize at run time, as in the Trace-Factored BigSwitch (TFB) of Park et al. [19], which halves the keys at the price of 50% more key switches and a whole switching error multiplied by the secret. Meanwhile, the Galois splitting of Park [20] (Eurocrypt’25) decomposes such switches into base-ring diagonals, but it has not been used to reduce their keys. However, whether a BigSwitch can publish only the keys of TFB and still run at the speed of the original one has remained open.

In this work, we first propose the *pair-merge BigSwitch*, which observes that the diagonals of a split BigSwitch are indexed by an inversion-closed coset κH and rewrites the target of a diagonal as $x g(s) + y s g(s) = g(g^{-1}(x) s + g^{-1}(y) s g^{-1}(s))$, so that it borrows the product key of its inverse at the cost of one Galois step; it makes the $2n$ key switches of the original BigSwitch with $n/2 + 1$ instead of n product keys, and we prove that no per-diagonal BigSwitch with $2n$ key switches holds fewer than $\frac{3}{2}n + 1$ keys. Building on it, we propose *public key rebasing*, which turns Galois keys into product keys with one relinearization key placed one small modulus layer above the chain, so that the client publishes only the keys of TFB and a 10.6 MiB layer key, at the price of $1.5\times$ the resident keys of TFB on the server. Consequently, together with a *Hermitian hop* that finishes a partner diagonal of a Gram product with one Galois step, PairSwitch makes $2n$ key switches on general inputs and $\frac{3}{2}n + 1$ on Gram products.

On the parameters of Park et al. [19] ($n = 256$, $p = 17$, $\log_2 PQ = 200$), our implementation runs a BigSwitch $1.89\text{--}2.00\times$ faster than TFB and $1.28\text{--}1.38\times$ faster than the original BigSwitch, with 2.4 bits less noise than TFB and 4.3 bits more than the original, with a layer key of 128.3 bits (default estimator model); it is $2.45\times$ faster than TFB on Gram inputs. Our implementation and the raw logs are public.¹

Keywords: Fully homomorphic encryption · Matrix-native FHE · Key switching · Galois automorphisms

¹ <https://github.com/acprk/pairswitch-gl>

1 Introduction

Fully homomorphic encryption (FHE) is a cryptographic primitive that allows one to compute on encrypted data without learning anything about it. Since Gentry’s first construction, the field has advanced steadily, and the RLWE-based schemes BGV [5] and CKKS [7] have become the schemes of choice for exact and approximate arithmetic. Among the computations that drive their deployment, matrix multiplication plays a central role: it dominates linear regression, the training of linear models, and the attention layers of transformer inference [13,22,20,3]. However, BGV and CKKS encrypt vectors rather than matrices, so an encrypted matrix product has to be emulated by many rotations and plaintext multiplications, and a long line of work has been devoted to reducing their number [13,22,20,3]. Recently, Gentry and Lee [10] proposed a different route, matrix-native FHE, in which a whole batch of matrices is encoded into one element of a multivariate ring and the product of two encrypted matrices costs only a few plain matrix products over $\mathbb{Z}_q$. Specifically, for a power of two n and an odd prime p , the Gentry–Lee (GL) scheme works over the base ring $R_q = \mathbb{Z}_q[i][X, W]/(X^n - i, \Phi_p(W))$ and the extended ring $R'_q = R_q[Y]/(Y^n - i)$, and it multiplies two encrypted matrices in the following four stages.

- **Encoding.** A batch of $\varphi(p)$ matrices $M^{(\ell)} \in M_n(\mathbb{C})$ is encoded into one element $m \in R'$ such that $m(\zeta_j, \zeta_k, \eta_\ell) \approx \Delta \cdot M_{jk}^{(\ell)}$ for a scaling factor Δ , where ζ_j and η_ℓ run over the primitive $4n$ -th and p -th roots of unity, the rows and columns being indexed by the X and Y roots.
- **Encryption.** With a secret $s \in R_q$ that does not involve Y , the matrix is encrypted as $(b, a) \in R_q'^2$ with $b + a s = m + e$, i.e. as n ordinary RLWE ciphertexts over R_q , one per power of Y .
- **Trace product.** For two ciphertexts (b_u, a_u) and (b_v, a_v) , GL compute $u \otimes v = \text{Tr}_Z(u(X, Z, W) \cdot \bar{v}(Y^{-1}, Z^{-1}, W^{-1}))$, where Tr_Z keeps the constant term in an auxiliary variable Z and the bar conjugates the Gaussian-integer coefficients; since s does not involve Z , this yields a four-component ciphertext

$$\text{Ct}_4 = (C_{00}, C_{10}, C_{01}, C_{11}) \quad \text{with} \quad C_{00} + C_{10} s + C_{01} s_Y + C_{11} s s_Y \approx u \otimes v,$$

where $s_Y = \bar{s}(Y^{-1}, W^{-1})$ is a conjugate copy of the secret in the Y variable. Each block costs a slot-wise matrix product over $\mathbb{Z}_q[i]$.

- **Relinearization $\text{Switch}_{\text{big}}$ and rescaling.** The four-component ciphertext is finally returned to a ciphertext under s alone. Since $C_{00} + C_{10} s$ is already under s , GL switch the two remaining terms with two key-switching keys over the extended ring,

$$\text{ksk}_{s_Y \rightarrow s} = \text{RLWE}'_s(q_o s_Y), \quad \text{ksk}_{s s_Y \rightarrow s} = \text{RLWE}'_s(q_o s s_Y) \in R_{qq_o}'^{2\ell},$$

where q_o is the special modulus and ℓ the number of gadget digits, and they output

$$(E_0, E_1) = (C_{00}, C_{10}) + \text{Switch}_{\text{big}}(C_{01}; \text{ksk}_{s_Y \rightarrow s}) + \text{Switch}_{\text{big}}(C_{11}; \text{ksk}_{s s_Y \rightarrow s}),$$

which decrypts under s to $C_{00} + C_{10}s + C_{01}s_Y + C_{11}s s_Y \approx u \otimes v$. Each $\text{Switch}_{\text{big}}$ decomposes its input into ℓ digits over R' , multiplies them with the ℓ rows of the key, and divides by q_o . Because the product carries the scale Δ^2 , a CKKS rescaling by Δ concludes the multiplication.

In this work we concentrate on this relinearization step, which Park et al. [19] call the BigSwitch: it holds the largest keys of the scheme and a large share of the time of a product. Every BigSwitch computes

$$(C_{01}, C_{11}) \mapsto C_{01} \cdot s_Y + C_{11} \cdot s s_Y \quad \text{under } s, \quad (1)$$

and its keys are defined over the extended ring R'_q , so that each of them is n times larger than an ordinary key. The original BigSwitch of Gentry and Lee switches the two terms of (1) with two such keys, of sources s_Y and $s s_Y$; this costs $2n$ base-ring key switches, but at $n = 256$ the two keys occupy 768 MiB. Recently, Park et al. [19] proposed the Trace-Factored BigSwitch (TFB): exploiting $(1, s, s_Y, s s_Y) = (1, s_Y) \otimes (1, s)$, it switches both terms with the key of source s_Y and then relinearizes $s^2 \rightarrow s$, which halves the keys to 385.5 MiB. However, this third switch per diagonal raises the cost to $3n$ base-ring key switches and acts on a ciphertext whose whole switching error is then multiplied by s . Park et al. [19] also amortize the extra cost over K accumulated Trace products, but each BigSwitch stays slower than the original. This raises the following question:

Can a BigSwitch publish only the keys of TFB and still run at the speed of the original one?

1.1 Our Contributions

In this work, we investigate the algebraic structure that the Galois group imposes on the BigSwitch of matrix-native FHE. We address the fundamental tension between the product keys of the original BigSwitch and the runtime relinearization of TFB through two symmetries of the problem: the inversion symmetry of the Galois automorphisms that index the diagonals, and the conjugate-transpose symmetry of the Trace product. Our main contributions are summarized below:

1. **Pair-merge BigSwitch with a provably optimal key set.** We introduce a novel construction that exploits the inversion symmetry of the coset κH that indexes the diagonals of a split BigSwitch. By establishing the pair-merge identity $x g(s) + y s g(s) = g(g^{-1}(x) s + g^{-1}(y) s g^{-1}(s))$, we show that a diagonal can borrow the product key of its inverse at the cost of one Galois step, so that $n/2 + 1$ product keys serve all n diagonals at the $2n$ key switches of the original BigSwitch, with no Y -axis transform and no switching error multiplied by s . We further prove that every per-diagonal BigSwitch with $2n$ key switches holds at least $\frac{3}{2}n + 1$ base-ring keys, so that the pairing is optimal among per-diagonal BigSwitches at this count, and confirm by an exhaustive search over all key sets at $n = 8$ that it lowers the minimal number of key switches for every key budget from 3 to 15 keys.

2. **Publicly derivable product keys.** We provide a key-rebasing procedure that derives every product key from a published Galois key and one relinearization key placed one small modulus layer above the chain, extending the server-side key derivation of [16,20,6] and the square-key multiplication of [14] to the quadratic sources $s \cdot g(s)$. Our approach identifies a non-prime layer modulus $P' = 2^{14} - 1$, evaluated exactly through an auxiliary NTT prime, as the key enabler that keeps the largest modulus at 128.3 bits of security under the default cost model of the lattice estimator (97.2 bits under classical core-SVP, against 107.3 for the chain itself). Consequently, PairSwitch publishes the Galois keys of TFB and a 10.6 MiB layer key, under a circular-security assumption of the same kind as that of TFB, and the derived keys add no measurable noise.
3. **Hermitian hop for structured products.** We identify the conjugate transpose γ as a second symmetry that maps every diagonal onto its inverse. By proving that $\mu_{t-1} \circ \gamma = g_t^{-1} \circ \mu_t$, we show that for γ -symmetric or antisymmetric phases a partner diagonal costs a single Galois step, which lowers Gram products, symmetrized sums, commutators and squaring chains to $\frac{3}{2}n + 1$ key switches and yields the conjugate transpose of any product with n further key switches and no second Trace product.
4. **Implementation and comprehensive evaluation.** We implement the BigSwitch of Gentry and Lee, TFB and PairSwitch in one RNS/NTT engine with identical limbs, digits and key format, and validate every output by exact decoding with negative controls. On the parameters of Park et al. [19], PairSwitch runs a BigSwitch $1.89\text{--}2.00\times$ faster than TFB with 2.4 bits less noise and $1.28\text{--}1.38\times$ faster than the original BigSwitch, as summarized in Table 1. Moreover, it is $2.45\times$ faster on Gram inputs, and it accelerates the GPT-2 attention kernel $O = PV$ of Park et al. by $1.35\times$ in our engine, squaring chains by $2.1\times$ on a non-128-bit five-limb chain, and shared-QK attention scores by $3.1\times$, or $2.3\times$ against a TFB that also exploits the symmetry.

These contributions play different roles. The splitting and the use of product keys instead of a runtime relinearization provide the online speed; key rebasing keeps the published keys at the size of TFB; the pairing lowers the resident keys by a quarter against the split direct route, at 5–6% of its time and at the cost of $1.5\times$ the resident keys of TFB; and the hop saves work on symmetric inputs.

Table 1. One BigSwitch on the parameters of Park et al. [19] ($n = 256$, $p = 17$, $\log_2 PQ = 200$). Keys are published and resident MiB, N counts base-ring key switches, times are medians in seconds, and ratios are paired against TFB.

scheme	keys		N	noise (bits)	time (s)		ratio	
	publ.	resid.			8 thr.	1 thr.	8 thr.	1 thr.
Gentry–Lee [10]	768.0	768.0	$2n$	15.37	0.929	6.557	0.691	0.667
TFB [19]	385.5	385.5	$3n$	22.07	1.345	9.844	1	1
PairSwitch	394.6	577.5	$2n$	19.67	0.680	4.957	0.506	0.501

1.2 Techniques Overview

Split form of the known BigSwitches. By the Galois splitting of Jai Hyun Park [20], the extended ring decomposes as $R'_q \cong \prod_{t \in H} R_q$ through maps μ_t with $\mu_t(s_Y) = \kappa \tau_t(s)$, where $H = \langle 5 \rangle$ is cyclic of order n . Writing

$$g_t := \kappa \tau_t, \quad x_t := n^{-1} \mu_t(C_{01}), \quad y_t := n^{-1} \mu_t(C_{11}),$$

the BigSwitch becomes n independent diagonals, and diagonal t must produce, under s ,

$$x_t g_t(s) + y_t s g_t(s). \quad (2)$$

The two known constructions produce (2) as follows:

- **Gentry–Lee:** $\text{KS}(x_t; K_{g_t(s)}) + \text{KS}(y_t; K_{s g_t(s)})$, i.e. two key switches and two keys per diagonal;
- **TFB:** $\text{KS}(x_t; K_{g_t(s)}) + s \cdot \text{KS}(y_t; K_{g_t(s)})$, where the factor s is removed by a runtime relinearization with K_{s^2} , i.e. three key switches, one key per diagonal, and a switching error multiplied by s .

Hence the former pays one product key per diagonal and the latter one runtime switch per diagonal. PairSwitch removes both costs with two insights.

Insight 1: inverse pairing. Since the complex conjugation $\kappa : (i, X, W) \mapsto (-i, X^{-1}, W^{-1})$ is an involution that commutes with H , the coset $\{g_t\}_{t \in H} = \kappa H$ is closed under inversion, $g_t^{-1} = g_{t^{-1}}$. Applying the ring automorphism g_t gives

$$x_t g_t(s) + y_t s g_t(s) = g_t \left(\underbrace{g_t^{-1}(x_t)}_{\text{key-free}} \cdot s + \underbrace{g_t^{-1}(y_t)}_{\text{partner's product key}} \cdot s g_{t^{-1}}(s) \right). \quad (3)$$

Therefore diagonal t is computed as follows, as shown on the right of Figure 1:

- the inner phase costs one key switch, with the product key of the partner t^{-1} ;
- the outer g_t costs one Galois step, with the linear key $K_{g_t(s)}$ that diagonal t holds anyway;
- the Galois step only permutes the error, so no error is multiplied by s .

Since H is cyclic, only 1 and $5^{n/2}$ are self-inverse, so $n/2 + 1$ representatives carry all product keys.

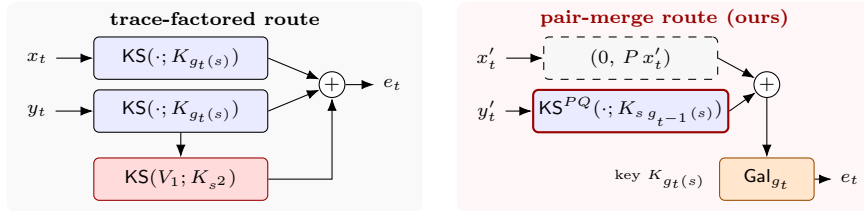


Fig. 1. Diagonal t in the trace-factored route and in the pair-merge route, with $x'_t = g_t^{-1}(x_t)$ and $y'_t = g_t^{-1}(y_t)$; the bottom box is the runtime step.

Insight 2: key rebasing. A product key is a rebased linear key, by the square-key multiplication of Kim et al. [14]. Indeed, for a row of $K_{g(s)}$ and a relinearization of its mask v ,

$$\begin{aligned} u + v s &= P g_h g(s) + e \quad \text{and} \quad a + b s = v s^2 + e' \\ \implies a + (u + b) s &= P g_h \cdot s g(s) + s e + e'. \end{aligned}$$

Hence $(a, u + b)$ is a row of $K_{s g(s)}$, obtained publicly from the published keys, and the derived product keys add nothing to the view of an adversary. The only requirement is that the relinearization key decomposes the full mask $v \in R_{PQ}$, so it lives one small modulus layer P' above PQ .

Putting the two insights together. The resulting PairSwitch works in three phases:

- **publish:** the client publishes the n Galois keys $K_{g_t(s)}$ and one relinearization key K'_{s^2} at modulus PQP' , i.e. the key set of TFB up to a small layer;
- **expand:** once, the evaluator rebases the linear keys of the $n/2 + 1$ representatives into product keys by Insight 2;
- **switch:** per call, it runs the direct route on the representatives and the pair merge (3) on their partners.

Consequently, PairSwitch makes $2n$ key switches with no Y -axis transform. In exchange, the noise carries one rounding per diagonal, about 4 bits whose impact depends on the precision budget of the computation, and the evaluator holds $\frac{3}{2}n + 1$ keys instead of the $n + 1$ of TFB. This is optimal among per-diagonal BigSwitches by Theorem 1.

2 Preliminaries

2.1 Notation

Table 2 summarizes the symbols used throughout. Logarithms are to base 2, and “noise of b bits” means an output error of ℓ_2 norm 2^b . For a modulus q , $[a]_q \in (-q/2, q/2]$ is the centered representative of a , applied coefficientwise to polynomials. Key sizes are in MiB of coefficient-form keys, 8 bytes per coefficient.

2.2 Cyclotomic Rings and Galois Groups

Let $n \geq 4$ be a power of two, let p be an odd prime, and let q be a modulus whose prime factors are all $\equiv 1 \pmod{4np}$. Following GL [10] and Park et al. [19], the base ring and the extended ring are

$$\begin{aligned} A_q &= \mathbb{Z}_q[i][W]/\Phi_p(W), & R_q &= A_q[X]/(X^n - i), \\ R'_q &= R_q[Y]/(Y^n - i), & R'_q &\cong R_q \otimes_{A_q} A_q[Y]/(Y^n - i). \end{aligned}$$

We write R, R' for the integral versions. Since $X^{4n} = 1$ and $i = X^n$, the base ring R is the ring of integers $\mathbb{Z}[\zeta_{4np}]$ of the $4np$ -th cyclotomic field and has $\mathbb{Z}$ -rank

Table 2. Summary of notation.

Symbol	Meaning	Symbol	Meaning
n, p	matrix size; odd prime	d	rank $2n\varphi(p)$ of R
R, R'	base ring; extended ring	s, s_Y	secret; copy $\gamma(s)$
κ, τ_t, γ	Galois maps	H, G	$\langle 5 \rangle; \langle \kappa \rangle \times H$
g_t	$\kappa\tau_t$	$\mathcal{E}$	representatives
Q, P, P'	moduli	L, k, ℓ	limbs; digits
K_σ	key $\sigma \rightarrow s$	$\text{KS}, \text{Gal}_\sigma$	key switch; Galois step
Ct_4, Φ	Trace product; phase	$\mu_t, n^{-1}\delta_t$	projection; idempotent
x_t, y_t	diagonal inputs	N, U	key switches; resident keys

$d = 2n\varphi(p)$, e.g. $d = 8192$ at $(n, p) = (256, 17)$. Moreover, R' is a free R -module with basis $1, Y, \dots, Y^{n-1}$; elements of $R \subset R'$ are called Y -free.

Slots and Galois group. When q is prime, the Chinese remainder theorem splits R_q into d copies of $\mathbb{Z}_q$, indexed by a branch $i \mapsto \pm\iota$ with $\iota^2 = -1$, a primitive $4n$ -th root of unity ζ with $\zeta^n = \pm\iota$ for X , and a root for W ; likewise, R'_q splits further along the roots of Y . Multiplication is therefore slot-wise, and the number-theoretic transforms (NTTs) along the X, Y and W axes convert between coefficients and slots. The Galois group of R over $\mathbb{Z}$ is $(\mathbb{Z}/4np)^\times$, which acts by $\zeta_{4np} \mapsto \zeta_{4np}^k$ and permutes the slots. Following GL [10, §3.6], we use the conjugation κ , the maps τ_t and the conjugate transpose $\gamma = \text{swap} \circ \kappa$, where swap exchanges X and Y :

$$\begin{aligned} \kappa : (i, X, Y, W) &\mapsto (-i, X^{-1}, Y^{-1}, W^{-1}), & \tau_t : X &\mapsto X^t, \\ \gamma : (i, X, Y, W) &\mapsto (-i, Y^{-1}, X^{-1}, W^{-1}). \end{aligned}$$

Since $X^n = i$ must be preserved, τ_t is defined exactly for $t \equiv 1 \pmod{4}$, and these t form the group

$$H := \{t \in (\mathbb{Z}/4n)^\times : t \equiv 1 \pmod{4}\} = \langle 5 \rangle,$$

which, for every power of two n , is cyclic of order n and generated by 5. Moreover, κ is an involution that commutes with every τ_t , so the automorphisms we use form the group $G := \langle \kappa \rangle \times H$. Consequently, the coset κH is closed under inversion,

$$(\kappa\tau_t)^{-1} = \kappa\tau_{t^{-1}}, \tag{4}$$

and, since H is cyclic of even order, exactly two of its elements are self-inverse, namely 1 and $5^{n/2}$. These two facts drive our construction.

2.3 RLWE Encryption and Hybrid Key Switching

Ciphertexts. Let $s \in R$ be a ternary secret, and let errors be discrete Gaussians of parameter $\sigma = 3.2$. A ciphertext over $S \in \{R, R'\}$ is a pair $(c_0, c_1) \in S_q^2$ with

$$\text{phase} \quad c_0 + c_1 s = \begin{cases} \Delta m + e & (\text{CKKS [7]}), \\ m + t_{\text{pt}} e & (\text{BGV [5]}), \end{cases}$$

for a small error e . More generally, an r -component ciphertext decrypts under a key vector $(1, \tau_1, \dots, \tau_{r-1})$ to $\sum_j c_j \tau_j$. Our constructions manipulate only such decryption equations and are therefore independent of the flavour [19]. Security rests on the RLWE assumption [17] and, for published evaluation keys, on the circular security discussed in Section 2.4.

RNS gadget. We use hybrid key switching [9,12]. The ciphertext modulus $Q = q_1 \cdots q_L$ has L word-size limbs, and P is a special modulus of k limbs. The limbs of Q are grouped into ℓ digits (“dnum”); for digit h with limb product Q_h ,

$$[c]_h = [c]_{Q_h} \text{ (centered residue)}, \quad g_h = (Q/Q_h) \cdot [(Q/Q_h)^{-1}]_{Q_h},$$

so that $c = \sum_{h < \ell} [c]_h g_h \pmod{Q}$ by the Chinese remainder theorem, since $g_h \equiv 1 \pmod{Q_h}$ and $g_h \equiv 0 \pmod{Q_{h'}}$ for $h' \neq h$.

Keys and key switching. A key $K_\sigma = K_{\sigma \rightarrow s}$ with source $\sigma \in S$ is a list of rows $(u_h, v_h) \in S_{PQ}^2$, $h < \ell$, generated with uniform v_h and fresh errors e_h such that

$$u_h + v_h s \equiv P g_h \sigma + e_h \pmod{PQ}.$$

Key switching computes

$$\text{KS}(c; K_\sigma) = \text{ModDown}_P \left(\sum_{h < \ell} \text{ModUp}([c]_h) \cdot (u_h, v_h) \right),$$

where ModUp extends a digit to all limbs of PQ and ModDown_P divides by P with rounding. The output has phase $c\sigma + \varepsilon$ with

$$\varepsilon = \underbrace{P^{-1} \sum_h [c]_h e_h}_{\text{key term}} + \underbrace{r_0 + r_1 s}_{\text{rounding term}},$$

where r_0, r_1 are the rounding errors of ModDown ; for a large P the rounding term dominates. We write KS_R for a key over R and $\text{KS}_{R'}$ for a key over R' ; a key over R acts on an R' -wide input slice-wise over Y [19]. We also write KS^{PQ} for a key switch whose output is left at modulus PQ , i.e. without the final ModDown . A key over R' consists of n keys over R , one per power of Y ; we therefore measure key sizes in base-ring keys, and one base-ring key occupies 1.5 MiB on the parameters of Park et al. [19].

2.4 Galois Automorphisms and Relinearization

For $\sigma \in G$, applying σ to a ciphertext (c_0, c_1) with phase φ under s yields phase $\sigma(\varphi)$ under $\sigma(s)$ [9]; a key switch then returns it under s . Similarly, the tensor of two ciphertexts under s has phase $d_0 + d_1s + d_2s^2$, and a key switch of d_2 removes the quadratic term [5]. Both are instances of one template: a source $\sigma(s)$ or $s \cdot g(s)$, with $g \in G$, is switched back to s by the corresponding key,

$$\begin{aligned} \text{Gal}_\sigma(c_0, c_1) &= (\sigma(c_0), 0) + \text{KS}_R(\sigma(c_1); K_{\sigma(s)}), & \text{phase } \sigma(\varphi) + \varepsilon, \\ \text{Relin}(d_0, d_1, d_2) &= (d_0, d_1) + \text{KS}_R(d_2; K_{s^2}), & \text{phase } d_0 + d_1s + d_2s^2 + \varepsilon. \end{aligned}$$

We call $K_{g(s)}$ a linear key and $K_{s \cdot g(s)}$ a product key; K_{s^2} is the product key of $g = \text{id}$, and product keys with $g \neq \text{id}$ appear in the transposition of ciphertext matrices [20] and in the BigSwitch. Since σ permutes the canonical embedding, a Galois step only permutes the input error and adds one switching error. For many Galois steps on one input, hoisting [11, 4] shares the **ModUp**, and baby-step/giant-step storage [11] reaches m automorphisms with about $2\sqrt{m}$ keys. Finally, the keys $K_{\sigma(s)}$ and K_{s^2} encrypt functions of s under s , so their security relies on the standard circular-security assumption that every CKKS, BGV or GL deployment with rotations already makes [9, 1]; keys with other sources, or several keys that are challenged jointly, are covered by explicit joint assumptions of the same kind, such as Assumption 3 of Park et al. [19].

2.5 Gentry–Lee Matrix Encryption and the Trace Product

We now recall how GL [10] pass from matrices to ciphertexts, and from two ciphertexts to the four-component ciphertext on which every BigSwitch acts.

From matrices to polynomials. For $\varphi(p)$ matrices $M^{(\ell)} \in M_n(\mathbb{C})$, the encoding is the unique $m \in R'$, up to rounding, with

$$m(\zeta_j, \zeta_k, \eta_\ell) = \Delta \cdot M_{jk}^{(\ell)}, \quad j, k \in [0, n), \ell \in [0, \varphi(p)),$$

where ζ_j runs over the n primitive $4n$ -th roots of unity with $\zeta_j^n = i$ and η_ℓ over the primitive p -th roots of unity. Hence the X roots index the rows, the Y roots the columns, and the W roots the batch.

From polynomials to ciphertexts. Since $s \in R$ does not involve Y , writing $m = \sum_{j < n} m_j Y^j$ with $m_j \in R$ gives

$$(b, a) = \left(\sum_{j < n} b_j Y^j, \sum_{j < n} a_j Y^j \right) \in R_q'^2, \quad b_j + a_j s = m_j + e_j,$$

i.e. n ordinary RLWE ciphertexts over R_q , one per power of Y . The second operand of a product is read in the Y variable, and its secret becomes the conjugate copy

$$s_Y := \gamma(s) = \bar{s}(Y^{-1}, W^{-1}),$$

which is the image of s under the public automorphism γ rather than an independent secret [10,19].

From two ciphertexts to a four-component ciphertext. GL multiply two encodings u, v by the trace operation [10, Def. 3.4], which keeps the constant term in an auxiliary variable Z with $Z^n = i$:

$$u \otimes v := \text{Tr}_Z(u(X, Z, W) \cdot \bar{v}(Y^{-1}, Z^{-1}, W^{-1})).$$

For ciphertexts (b, a) of u and (β, α) of v , the secrets do not involve Z and can be taken out of the trace, so that

$$(b + as) \otimes (\beta + \alpha s) = \Phi := C_{00} + C_{10} s + C_{01} s_Y + C_{11} s s_Y$$

with

$$C_{00} = b \otimes \beta, \quad C_{10} = a \otimes \beta, \quad C_{01} = b \otimes \alpha, \quad C_{11} = a \otimes \alpha.$$

Following Park et al. [19], we call this the Trace product, and we denote by $\text{Ct}_4 = (C_{00}, C_{10}, C_{01}, C_{11})$ the resulting four-component ciphertext with phase Φ . It decodes to $n^{-1}M_u M_v^H$ per W -slot, and each of the four blocks costs one slot-wise matrix product over $\mathbb{Z}_q[i]$ [10, Thm. 3.6]. Because the product is bilinear and the key vector is fixed, K Trace products can also be summed in Ct_4 form before a single BigSwitch, as in the FRee-L technique of Park et al. [19].

3 Previous BigSwitch Constructions

This section reviews the two BigSwitch constructions on which we build, namely the original one of Gentry and Lee [10] and the trace-factored one of Park et al. [19], together with the Galois splitting of Jai Hyun Park [20]. Together, they form the baseline for our construction in Section 4.

Definition 1 (BigSwitch [19]). *A BigSwitch is an algorithm that, on input $\text{Ct}_4 = (C_{00}, C_{10}, C_{01}, C_{11}) \in R_q'^4$, outputs a pair $(E_0, E_1) \in R_q'^2$ with*

$$E_0 + E_1 s = \Phi + e_{\text{BS}} = C_{00} + C_{10} s + C_{01} s_Y + C_{11} s s_Y + e_{\text{BS}},$$

where the error e_{BS} lies within the decryption radius.

3.1 The Gentry–Lee BigSwitch

The original construction of Gentry and Lee [10, §3.5] switches the two terms that are not yet under s separately, each with its own key over the extended ring. Specifically, it uses the two keys

$$K_{s_Y} = \{(u_h, v_h)\}_{h < \ell}, \quad K_{s s_Y} = \{(u'_h, v'_h)\}_{h < \ell} \subset R_{PQ}'^2,$$

with $u_h + v_h s \equiv P g_h s_Y + e_h$ and $u'_h + v'_h s \equiv P g_h s s_Y + e'_h$, and it runs Algorithm 1. Each of the two key switches decomposes an R' -wide input into ℓ digits, so that every ModUp and ModDown transforms all n slices along the X, W and Y axes. As a result, the Gentry–Lee BigSwitch makes $2n$ base-ring key switches and its error consists of two ordinary switching errors. However, it publishes two keys over R' , i.e. $2n$ base-ring keys, which occupy 768 MiB at $n = 256$.

Algorithm 1 Gentry–Lee BigSwitch [10]

Require: $\text{Ct}_4 = (C_{00}, C_{10}, C_{01}, C_{11}) \in R_q^4$; keys $K_{s_Y} = \{(u_h, v_h)\}_h$ and $K_{s s_Y} = \{(u'_h, v'_h)\}_h$ over R'
Ensure: $(E_0, E_1) \in R_q^2$ with $E_0 + E_1 s \approx \Phi$
1: **for** $h = 0, \dots, \ell - 1$ **do**
2: $d_h \leftarrow \text{ModUp}([C_{01}]_h)$, $d'_h \leftarrow \text{ModUp}([C_{11}]_h)$ $\triangleright R'$ -wide digits, all axes
3: **end for**
4: $(A_0, A_1) \leftarrow \sum_{h < \ell} d_h \cdot (u_h, v_h) + \sum_{h < \ell} d'_h \cdot (u'_h, v'_h)$ $\triangleright$ accumulate at PQ
5: $(A_0, A_1) \leftarrow \text{ModDown}_P(A_0, A_1)$ $\triangleright$ phase $C_{01}s_Y + C_{11}s s_Y + \text{error}$
6: **return** $(C_{00} + A_0, C_{10} + A_1)$

3.2 The Trace-Factored BigSwitch

Park et al. [19] observe that the key vector of a Trace product is a tensor product,

$$(1, s, s_Y, s s_Y) = (1, s_Y) \otimes (1, s),$$

and therefore route the product secret $s s_Y$ through two cheaper keys. Specifically, they switch C_{11} with the key of source s_Y instead of $s s_Y$, which yields a ciphertext (V_0, V_1) with

$$V_0 + V_1 s \approx C_{11} s_Y \quad \implies \quad V_0 s + V_1 s^2 \approx C_{11} s s_Y,$$

so the remaining quadratic term $V_1 s^2$ is an ordinary relinearization over the base ring, applied to each of the n Y -slices of V_1 with the same key K_{s^2} . Algorithm 2 summarizes the resulting procedure. Its design has three consequences.

- **Keys.** TFB never generates the product-secret key $K_{s s_Y}$; it publishes one key over R' and one relinearization key, i.e. $n + 1$ base-ring keys (385.5 MiB at $n = 256$), which saves exactly a fraction $(n - 1)/(2n)$ of the keys of Gentry and Lee.
- **Cost and noise.** The relinearization is a third key switch on every diagonal, so TFB makes $3n$ base-ring key switches, n of them on the runtime input V_1 . Moreover, the switching error e_V of (V_0, V_1) is multiplied by s in $V_0 s + V_1 s^2$; in our reproduction this term dominates and costs about +6.7 bits over the Gentry–Lee BigSwitch at $n = 256$. To amortize the extra switch, Park et al. [19] accumulate K Trace products in Ct_4 form before one BigSwitch (Free-L), which reduces the relative overhead as $O(1/K)$ but leaves the cost of each BigSwitch unchanged.
- **Assumption.** The two published keys have the related sources s_Y and s^2 , so the pseudorandomness of the pair rests on a joint related-secret assumption, which Park et al. [19] state as their Assumption 3.

3.3 The Galois Splitting of Park

Recently, Jai Hyun Park [20, §3.1, Alg. 2] decomposed a key switch toward a transposed secret into ordinary automorphism key switches between a pre- and

Algorithm 2 Trace-Factored BigSwitch (TFB) [19]

Require: $\text{Ct}_4 = (C_{00}, C_{10}, C_{01}, C_{11}) \in R_q'^4$; key $K_{s_Y} = \{(u_h, v_h)\}_h$ over R' ; relin-earization key K_{s^2} over R

Ensure: $(E_0, E_1) \in R_q'^2$ with $E_0 + E_1 s \approx \Phi$

```

1: for  $h = 0, \dots, \ell - 1$  do
2:    $d_h \leftarrow \text{ModUp}([C_{01}]_h), d'_h \leftarrow \text{ModUp}([C_{11}]_h)$   $\triangleright R'$ -wide digits, all axes
3: end for
4:  $(U_0, U_1) \leftarrow \text{ModDown}_P(\sum_h d_h \cdot (u_h, v_h))$   $\triangleright$  phase  $C_{01} s_Y$ 
5:  $(V_0, V_1) \leftarrow \text{ModDown}_P(\sum_h d'_h \cdot (u_h, v_h))$   $\triangleright$  same key: phase  $C_{11} s_Y$ 
6: for  $j = 0, \dots, n - 1$  do  $\triangleright$  runtime, per  $Y$ -slice
7:    $(W_0^{(j)}, W_1^{(j)}) \leftarrow \text{KS}_R(V_1^{(j)}; K_{s^2})$   $\triangleright V_1^{(j)} s^2 \rightarrow s$ 
8: end for
9:  $E_0 \leftarrow C_{00} + U_0 + W_0, E_1 \leftarrow C_{10} + U_1 + V_0 + W_1$   $\triangleright V_0 s$  joins the  $s$ -coefficient
10: return  $(E_0, E_1)$ 

```

a post-processing map. In GL, this decomposition takes the following form. For $t \in H$, define the A_q -algebra maps and elements

$$\mu_t : R_q' \rightarrow R_q, \quad X \mapsto X, \quad Y \mapsto X^t; \quad \delta_t := \sum_{m=0}^{n-1} X^{-mt} Y^m \in R_q',$$

where μ_t is well defined because $(X^t)^n = i$.

Lemma 1 (Galois splitting [20]). *Let every prime factor of q be $\equiv 1 \pmod{4np}$. Then the following hold.*

- (a) $\mu_t(\delta_{t'}) = n \cdot [t = t']$ for all $t, t' \in H$.
- (b) $\mu = (\mu_t)_{t \in H} : R_q' \rightarrow \prod_{t \in H} R_q$ is an isomorphism of R_q -algebras with inverse $(x_t)_t \mapsto n^{-1} \sum_t \delta_t x_t$.
- (c) $\mu_t|_{R_q} = \text{id}$ and $\mu_t \circ \gamma = \kappa \tau_t$ on R_q . Consequently, $\mu_t(s_Y) = \kappa \tau_t(s)$ and $\mu_t(s s_Y) = s \cdot \kappa \tau_t(s)$.

The proof is given in Appendix A; part (b) is the ring-theoretic form of Park's identity $N' \cdot m_j = \sum_t X^{-j(2t+1)} m(X^{2t+1})$ for the ring degree N' of [20, §3.1]. Applying Lemma 1(b) and (c) to the two terms of (1) yields, for every Ct_4 ,

$$C_{01} s_Y + C_{11} s s_Y = \sum_{t \in H} \delta_t (x_t \kappa \tau_t(s) + y_t s \kappa \tau_t(s)), \quad (5)$$

where $x_t = n^{-1} \mu_t(C_{01})$ and $y_t = n^{-1} \mu_t(C_{11})$. In other words, the target of a BigSwitch decomposes into n independent diagonals with the targets (2), recombined by the elements δ_t , i.e. by the idempotents $n^{-1} \delta_t$. Note that n^{-1} must be applied to the public input, because $n^{-1} \delta_t$ has coefficients of size about q and would destroy any error that it multiplies [20, Alg. 2].

4 Galois-Paired BigSwitch

In the previous section, we recalled that both known BigSwitches are R' -wide and that the Galois splitting decomposes them into n diagonals. We now run

these diagonals as independent base-ring pipelines, which removes the Y -axis transforms, and then pair them, which removes the third key switch of TFB.

4.1 Galois-Diagonal BigSwitch

In Algorithms 1 and 2, every key switch is R' -wide, so each **ModUp** and **ModDown** transforms n slices along the X and W axes and, in addition, along the Y axis. Since these transforms take 79–86% of a key switch on one core, against 4–8% for the multiply–accumulate with the key, the Y -axis share is worth removing. In the RLWE setting of Park [20], the maps around the splitting cost $\tilde{O}(N'^2)$ for the ring degree N' . In GL, by contrast, the Y axis has its own transform, and in the evaluation domain the two maps of Lemma 1 act as follows:

- μ_t gathers the slots $\{(a, b) : b = a^t\}$, one R -element of slots per diagonal;
- multiplication by δ_t scatters them back with a factor n .

Both are index maps, identical for every RNS prime. Consequently, by (5), the BigSwitch reduces to n independent R -sized ciphertexts e_t under s with phase $x_t g_t(s) + y_t s g_t(s)$,

$$(E_0, E_1) = (C_{00}, C_{10}) + \sum_{t \in H} \delta_t e_t, \quad (6)$$

each of which runs as a pipeline of **ModUp**, multiply–accumulate and **ModDown** without any Y -axis transform. Algorithm 3 states this Galois-diagonal BigSwitch for an arbitrary route per diagonal. With the trace-factored route on every diagonal, it uses exactly the keys of TFB by Proposition 1 below, and we call it GDB-TFB. It is a direct instantiation of the splitting of Park [20] in GL, and it already takes $0.68\text{--}0.73\times$ the time of TFB.

Algorithm 3 Galois-Diagonal BigSwitch

Require: $\text{Ct}_t \in R_q^{r_A}$, for every $t \in H$, a route and its keys over R

Ensure: $(E_0, E_1) \in R_q^{r_2}$ with $E_0 + E_1 s \approx \Phi$

```

1: for all  $t \in H$  in parallel do
2:    $x_t \leftarrow n^{-1} \mu_t(C_{01}), y_t \leftarrow n^{-1} \mu_t(C_{11})$  ▷ slot gather, no transform
3:   if the route of  $t$  is trace-factored then
4:      $(U_0, U_1) \leftarrow \text{KS}_R(x_t; K_{g_t(s)}), (V_0, V_1) \leftarrow \text{KS}_R(y_t; K_{g_t(s)})$  ▷  $R$ -sized
5:      $(W_0, W_1) \leftarrow \text{KS}_R(V_1; K_{s^2})$  ▷ runtime relinearization
6:      $e_t \leftarrow (U_0 + W_0, U_1 + V_0 + W_1)$ 
7:   else if the route of  $t$  is direct then
8:      $e_t \leftarrow \text{ModDown}_P(\text{KS}^{PQ}(x_t; K_{g_t(s)}) + \text{KS}^{PQ}(y_t; K_{s g_t(s)}))$ 
9:   else ▷ pair-merge route of Section 4.2
10:     $e_t \leftarrow$  Algorithm 4 on  $(x_t, y_t)$ 
11:   end if
12: end for
13: return  $(C_{00}, C_{10}) + \sum_{t \in H} \delta_t e_t$  ▷ slot scatter  $\times n$ , no transform

```

Noise of the split form. Let v_k and v_r be the per-coefficient variances of the key and rounding terms of one R -sized switch. Since the scatter puts one

signed coefficient of each e_t into every output coefficient, the output variance per switched term is

$$\underbrace{n(v_k + v_r)}_{\text{one ModDown per diagonal}} \quad \text{versus} \quad \underbrace{nv_k + v_r}_{R'\text{-wide switch}},$$

so that, when rounding dominates, the split form costs $+\frac{1}{2}\log_2 n$ bits; we measured +2.00, +3.00 and +4.00 bits at $n = 16, 64, 256$. A following rescale shrinks both errors alike, so whether the gap matters depends on the precision budget of the computation.

Big keys and Galois keys. Finally, the splitting relates the key of TFB to the keys of the diagonal pipelines.

Proposition 1 (Galois decomposition of a big key). *Let K_{s_Y} be any key with source s_Y over R' . For $t \in H$, $K_{\kappa\tau_t} := \sum_m X^{tm} \cdot \text{slice}_m(K_{s_Y})$ is a key $\kappa\tau_t(s) \rightarrow s$, and the map from the slices to $(K_{\kappa\tau_t})_{t \in H}$ is invertible with inverse $n^{-1} \sum_t X^{-tm}$. For a fresh K_{s_Y} , the image has uniform masks and Gaussian errors of standard deviation $\sqrt{n}\sigma$, but its joint error is correlated.*

The proof is Lemma 1(b) in coefficient form. Hence the n Galois keys $K_{\kappa\tau_t(s)}$ carry exactly the bytes of the big key of TFB. With discrete errors, however, the converted tuple lies on a sublattice that fresh Galois keys hit with probability 0, so Appendix C relates their security in both directions.

4.2 Our Novel Pair-Merge Construction

GDB-TFB still makes three key switches per diagonal. We now remove the third one by exploiting the group structure of the Galois automorphisms that index the diagonals: the inversion symmetry of this group pairs the diagonals, and one product key then serves both diagonals of a pair at the price of one Galois step.

Motivation: beyond the two known routes. The direct route holds one product key per diagonal, whereas the trace-factored route replaces it by a runtime relinearization whose error is multiplied by s . Both treat the n diagonals independently. Our goal is to keep the two key switches per diagonal of the direct route while holding product keys on only half of the diagonals.

Key observation: the Galois group pairs the diagonals. We index the diagonals by exponents, $t = 5^e$ with $e \in \mathbb{Z}/n$, and write $g_t := \kappa\tau_t$. By (4), $g_t^{-1} = g_{t^{-1}}$ with $t^{-1} = 5^{-e}$, and $e \mapsto -e$ is an involution of $\mathbb{Z}/n$ with the two fixed points 0 and $n/2$. We therefore call

$$\mathcal{E} := \{5^e : 0 \leq e \leq n/2\}$$

the representatives, $n/2 + 1$ of them, and the other $n/2 - 1$ elements of H the partners, so that every partner t has $t^{-1} \in \mathcal{E}$, as illustrated in Figure 2. For instance, at $n = 16$ the representatives are $5^0, \dots, 5^8$. The partner $5^{-3} = 5^{13}$ borrows the product key $K_{s_{g_{5^3}(s)}}$ of its representative 5^3 , while 1 and 5^8 are

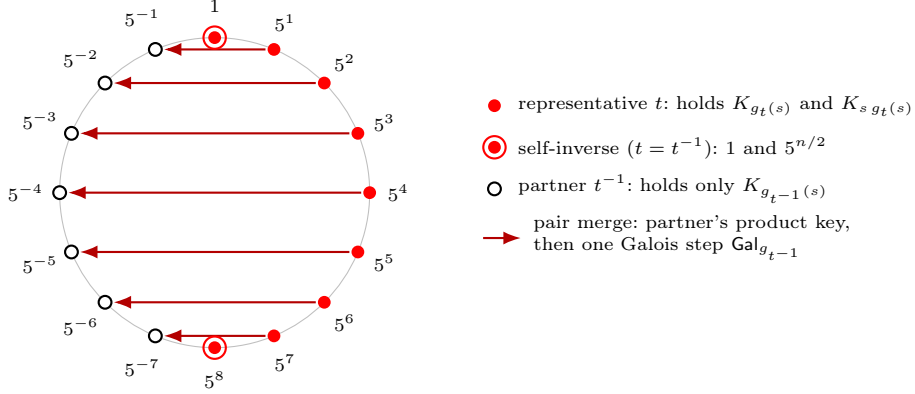


Fig. 2. Inverse pairing of the $n = 16$ diagonals $t = 5^e$; filled points are representatives.

their own inverses, so that $9 = n/2 + 1$ product keys serve all 16 diagonals. Accordingly, PairSwitch holds the n linear keys $K_{g_t(s)}$ for all $t \in H$, but the product keys $K_{s g_t(s)}$ only for $t \in \mathcal{E}$; Figure 3 compares this key set with those of the two known routes. A representative runs the direct route, whereas a partner runs the pair-merge route below, which uses the product key of its representative.

Lemma 2 (pair merge). *Let $t \in H$, $g = g_t$ and $x, y \in R_q$. Then*

$$x g(s) + y s g(s) = g\left(g^{-1}(x) \cdot s + g^{-1}(y) \cdot s g^{-1}(s)\right), \quad g^{-1} = g_{t^{-1}}.$$

Proof. g is a ring automorphism of R_q , so the right-hand side is $x g(s) + y g(s) s$.

The identity is elementary; what matters is the keys that its inner phase calls for, namely two sources:

- the source s , which is free on a public input, since $g^{-1}(x) \cdot s$ is the phase of $(0, g^{-1}(x))$; and
- the source $s g_{t^{-1}}(s)$, which is the quadratic source of diagonal t^{-1} .

Hence the product key of one diagonal of a pair $\{t, t^{-1}\}$ serves both, and the remaining work is one Galois step with the linear key $K_{g_t(s)}$ that diagonal t needs anyway. Since a Galois step acts on the phase by an automorphism, the switching error of the inner key is only permuted, and no switching error is multiplied by s . Algorithm 4 spells out the route; in particular, the term $P g_t^{-1}(x_t)$ has P -residue 0, so adding it at PQ is exact, and only the mask Z_1 is brought down to Q before the Galois step, while Z_0 stays at PQ and is permuted by g_t , an index map.

Proposition 2 (correctness of the merge route). *For all public x_t, y_t , Algorithm 4 returns an R -ciphertext under s with phase $x_t g_t(s) + y_t s g_t(s) + \varepsilon$, where $\varepsilon = g_t(\varepsilon_1 - r_1 s) + \varepsilon_2$. Here ε_1 is the key error of the product-key switch divided by P , $r_1 = P^{-1}[Z_1]_P$ is the rounding of the mask in line 4, and ε_2 is the error of the Galois step and of the final ModDown.*

Algorithm 4 Pair-merge route for a partner diagonal t , with $t^{-1} \in \mathcal{E}$

Require: public inputs $x_t = n^{-1}\mu_t(C_{01})$ and $y_t = n^{-1}\mu_t(C_{11})$; the partner's product key $K_{s_{g_{t-1}}(s)}$; the linear key $K_{g_t(s)}$

Ensure: $e_t \in R_q^2$ under s with phase $x_t g_t(s) + y_t s g_t(s) + \text{error}$

- 1: $x'_t \leftarrow g_t^{-1}(x_t), y'_t \leftarrow g_t^{-1}(y_t)$ ▷ index maps on every limb
 - 2: $(Z_0, Z_1) \leftarrow \text{KS}^{PQ}(y'_t; K_{s_{g_{t-1}}(s)})$ ▷ partner's product key, kept at PQ
 - 3: $Z_1 \leftarrow Z_1 + P \cdot x'_t$ ▷ key-free term $x'_t \cdot s$; exact at PQ
 - 4: $c \leftarrow g_t(\text{ModDown}_P(Z_1))$ ▷ only the mask goes down to Q
 - 5: $(A_0, A_1) \leftarrow (g_t(Z_0), 0) + \text{KS}^{PQ}(c; K_{g_t(s)})$ ▷ one runtime Galois step
 - 6: **return** $e_t \leftarrow \text{ModDown}_P(A_0, A_1)$
-

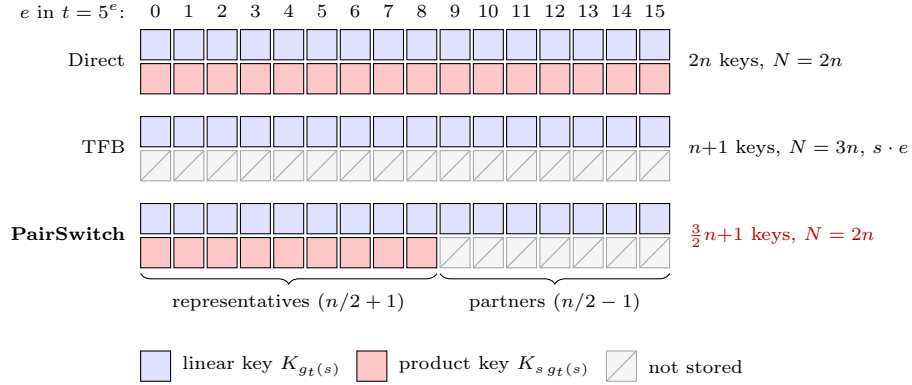


Fig. 3. Keys stored per diagonal for $n = 16$.

Proof. After line 3, Z has phase $P(x'_t s + y'_t s g_{t-1}(s)) + P\varepsilon_1$ modulo PQ . Line 4 replaces Z_1 by $Z_1 - [Z_1]_P$ up to the factor P , which subtracts $[Z_1]_P s = P r_1 s$ from the phase, and line 5 completes the Galois step Gal_{g_t} , which maps a phase φ to $g_t(\varphi)$ plus its switching error. By Lemma 2 and since $g_t^{-1} = g_{t-1}$, g_t maps the inner phase to $x_t g_t(s) + y_t s g_t(s)$. Line 6 divides by P . Here $r_1 s$ is the rounding term of hybrid key switching; no switching error is multiplied by s .

With the direct route on the representatives and Algorithm 4 on the partners, Algorithm 3 becomes PairSwitch.

4.3 Cost and Noise Analysis

Figure 3 summarizes the keys that each route holds per diagonal for $n = 16$. The Gentry–Lee route holds a linear and a product key on every diagonal, i.e. $2n$ keys; TFB holds only the n linear keys and one relinearization key, and pays for it with a third key switch on every diagonal; PairSwitch holds the n linear keys and the product keys of the $n/2 + 1$ representatives only, i.e. $\frac{3}{2}n + 1$ keys, and keeps two key switches per diagonal. The following proposition makes the resulting cost exact.

Proposition 3 (cost). *Count NTTs of one RNS limb of one R element, and let $M_{\text{up}} = \ell(L + k)$ and $M_{\text{dn}} = L + k$ be the counts of one ModUp and of the ModDown of one component. Per BigSwitch, PairSwitch makes $2n$ R -sized key applications, $\frac{3}{2}n + 1$ of them on public inputs and $n/2 - 1$ (the Galois steps of the partners) at run time; it performs*

$$(n/2 + 1)(2M_{\text{up}} + 2M_{\text{dn}}) + (n/2 - 1)(2M_{\text{up}} + 3M_{\text{dn}}) = (L + k)[(2\ell + 2.5)n - 1]$$

such NTTs and no Y -axis transform, and it holds $\frac{3}{2}n + 1$ base-ring keys.

Proof. A representative runs two ModUps and one two-component ModDown, its two public-input switches being summed at PQ . A partner runs two ModUps (one per key switch) and three component ModDowns (the mask before the Galois step and the final pair). Gather, scatter and g_t are index maps.

On the chain of Park et al. [19] ($L = \ell = 3$, $k = 1$, $n = 256$) this is 8700 NTTs, against 8192 for the split direct route and 12288 for TFB; our executor counts exactly these numbers. Relative to the R -sized GDB-TFB the model predicts $(2\ell + 2.5)/(3\ell + 3) = 0.708$ of the time. Two further constant-factor levers do not pay: streaming the shared product key of a pair once for both of its uses changes the time by 0.2% (the MAC is 5.7% of the thread time and the NTTs 84.6%), and fusing the gathers with the pre-twists makes it 0.5% slower.

Noise. Both routes end every diagonal with its own ModDown, so PairSwitch has the per-diagonal rounding of the splitting described in Section 4.1. By Proposition 2, the merge route adds one ordinary switching error per partner and nothing multiplied by s . We measure 19.67 bits, against 19.37 for the split direct route, 15.37 for the R' -wide Gentry–Lee BigSwitch, 22.07 for TFB and 26.08 for GDB-TFB. The $\frac{1}{2} \log_2 n \approx 4$ -bit gap to the R' -wide Gentry–Lee BigSwitch is thus the price of the splitting, whose impact depends on the precision budget.

Low-noise mode. When the output is not rescaled, every diagonal can scatter the mask component at PQ and leave its ModDown to one R' -wide pass, as an R' -wide switch does; only this component matters, because its rounding is the one multiplied by s . This lowers the noise to 18.89 bits at 1.04 – $1.09\times$ the time, with the same keys. Removing the partners' mid-route ModDown as well reaches the 15.4 bits of the R' -wide Gentry–Lee BigSwitch only with fresh product keys and 704.5 MiB of upload; with derived keys the floor is 16.5 bits at $+17$ – 20% time. We therefore keep per-diagonal ModDown as the default.

4.4 Optimality of the Pairing

PairSwitch reaches the $2n$ key switches of the direct method with $\frac{3}{2}n + 1$ resident base-ring keys. We now show that no BigSwitch of the same shape does better at this count. A per-diagonal BigSwitch computes every diagonal t with its own straight-line program on the public inputs x_t, y_t , in which

- a key application decomposes any value, public or derived from earlier outputs, against a key whose source is $h(s)$ or $h_1(s)h_2(s)$ for automorphisms h, h_1, h_2 ;

- ciphertexts are combined only with small integer coefficients and automorphisms, and public data enter as key-free ciphertexts (d_0, d_1) or as key-application inputs.

This model, stated precisely in Appendix D, contains Gentry–Lee, TFB, GDB-TFB, PairSwitch and the memory-constrained key sets of Section 5.2. It ignores noise, treats the mask of every key application as a fresh uniform value, and requires correctness for every $s \in R_q$, which all these constructions satisfy. It excludes the Hermitian hop of Section 5.2, which reuses the output of another diagonal and is correct only on symmetric inputs; this is why the hop makes fewer than $2n$ key applications.

Theorem 1 (optimality of the pairing). *Every per-diagonal BigSwitch makes at least $2n$ key applications. If it makes exactly $2n$, then every diagonal t uses the linear key $K_{g_t(s)}$ and a product key of source $sg_t(s)$ or $sg_{t^{-1}}(s)$. Consequently, it holds at least $n + (n/2 + 1) = \frac{3}{2}n + 1$ base-ring keys, and PairSwitch attains this bound.*

The proof, given in Appendix D, shows that a diagonal with two key applications either runs the direct route or reuses its first output through a Galois step, and that the latter forces exactly the source of the inverse diagonal; hence pairing is the only way to save product keys at this count. The theorem fixes one point of the trade-off between resident keys U and key applications N . At $n = 8$, an exhaustive search over all 2^{31} key sets, with per-diagonal costs that a shortest-path search over all monomial programs confirmed in 2760 random cases, gives the whole frontier: pairing lowers the minimal N at every U from 3 to 15, by up to three applications, e.g. $U = 13$ gives $N = 16 = 2n$ against 19 without it. Beyond the per-diagonal model, a dimension argument in Appendix E shows $U + W_0 + 2W_C \geq 2n$ for every BigSwitch, where W_0 and W_C count the runtime switches whose inputs do not and do depend on Ct_4 ; in particular $U \geq 2n$ when no switch is applied at run time, so the direct method is optimal there.

Remark 1 (pairing and the factor 2). The factor 2 on W_C cannot be dropped. A runtime output may serve different targets on different inputs: the Galois step of a merge route supplies the linear target when $C_{01} \neq 0$ and the quadratic one when $C_{11} \neq 0$. With symmetric baby-step/giant-step sets and one paired product key per pair, described in Section 5.2, this gives $U + W = \frac{3}{2}n + O(\sqrt{n})$; at $n = 256$ our engine executes $(U, W) = (162, 246)$, so $U + W = 408 < 2n$.

5 PairSwitch with Publicly Expanded Keys

The construction of Section 4 still relies on product keys that the client would have to publish. We now remove them from the public key set, extend the construction to structured inputs, and integrate both into PairSwitch.

5.1 Public Expansion of Product Keys

Publishing the product keys would cost $n/2 + 1$ base-ring keys on top of the key set of TFB and add a new kind of key to the public tuple. Instead, the evaluator derives them, following server-side key derivation [16,20,6] and the square-key multiplication of LFHE [14]. Fix a representative $t \in \mathcal{E}$, write $g = g_t$, and let $(u, v) \in R_{PQ}^2$ be row h of the published linear key, so that

$$u + v s = P g_h g(s) + e.$$

The evaluator turns this row into row h of $K_{sg(s)}$ in three steps.

- **Step 1 (decompose the mask).** Since v is a full element of R_{PQ} , special limb included, it is split into 12-bit digits, $v = \sum_{j < 17} 2^{12j} v_j$.
- **Step 2 (relinearize the mask).** With the layer key $K'_{s^2} = \{(u'_j, v'_j)\}_{j < 17}$ at modulus PQP' ,

$$(a, b) = \text{ModDown}_{P'} \left(\sum_{j < 17} \text{ModUp}(v_j) \cdot (u'_j, v'_j) \right), \quad a + b s = v s^2 + e'.$$

- **Step 3 (rebase).** Adding the old body to the new mask gives

$$a + (u + b) s = s(u + v s) + e' = P g_h \cdot s g(s) + s e + e',$$

so that $(a, u + b)$ is row h of a product key of source $sg(s)$.

Step 2 decomposes all of PQ , so the layer key needs its own special modulus P' above PQ ; its parameters are chosen as follows.

- **Layer modulus.** P' is a special modulus and never an NTT modulus of the ciphertexts, so it need not be prime; we take $P' = 2^{14} - 1$ and evaluate its limb exactly through an auxiliary 60-bit NTT prime, since all intermediate sums stay below 2^{44} . The key modulus is then $\log_2(PQP') < 214$ bits, for which the lattice estimator [2,15] gives 128.3 bits of security at $d = 8192$ under its default cost model; a layer of 15 bits would give only 127.5 bits. Appendix F lists the estimates.
- **Digits.** The 17 rows of 12-bit digits make the layer key 10.62 MiB, or $\rho = 7.08$ base-ring keys, and it is used only during the expansion.
- **Alternatives.** Without a layer, a key at modulus PQ with whole-limb digits leaves an error of $2^{92.5}$ and makes the derived key useless, while a pure power-of-two gadget over PQ works but costs 51–100 MiB and +0.4–0.8 bits.

Cost and noise. Algorithm 5 runs $(n/2 + 1)\ell$ base-ring key switches once per key set, i.e. 387 switches at $n = 256$, which take about 4s on 8 threads and 27s on one core; against TFB, this setup pays for itself after about five BigSwitch calls. The derived row error $se + e'$ is about $2^{8.4}$ per coefficient instead of $2^{1.7}$ for a fresh row, but a BigSwitch divides the key error by P , so the output noise is 19.67 bits with derived and 19.66 bits with fresh product keys.

Steps 1–3 prove the following lemma, which we record for later reference.

Lemma 3 (key rebasing, cf. [14]). *Let $(u, v) \in R_{PQ}^2$ satisfy $u + v s = P g_h g(s) + e$, and let $(a, b) \in R_{PQ}^2$ satisfy $a + b s = v s^2 + e'$. Then $a + (u + b) s = P g_h \cdot s g(s) + s e + e'$.*

Algorithm 5 Public expansion of the product keys

Require: linear keys $K_{g_t(s)} = \{(u_{t,h}, v_{t,h})\}_{h < \ell}$ for $t \in \mathcal{E}$; layer key $K'_{s^2} = \{(u'_j, v'_j)\}_{j < 17}$ at modulus PQP'

Ensure: product keys $K_{s g_t(s)}$ for all $t \in \mathcal{E}$, at modulus PQ

```

1: for all  $t \in \mathcal{E}$  and  $h < \ell$  in parallel do
2:    $(u, v) \leftarrow (u_{t,h}, v_{t,h})$   $\triangleright u + v s = P g_h g_t(s) + e$ 
3:    $(v_j)_{j < 17} \leftarrow$  12-bit digits of  $v \in R_{PQ}$   $\triangleright$  full mask,  $P$ -limb included
4:    $(a, b) \leftarrow \text{ModDown}_{P'}(\sum_{j < 17} \text{ModUp}(v_j) \cdot (u'_j, v'_j))$   $\triangleright a + b s = v s^2 + e'$ 
5:   row  $h$  of  $K_{s g_t(s)} \leftarrow (a, u + b)$   $\triangleright$  Step 3
6: end for
7: return  $(K_{s g_t(s)})_{t \in \mathcal{E}}$   $\triangleright$  the layer key may now be deleted

```

5.2 Extensions to Structured Products

The pair merge treats every input in the same way. Many matrix workloads, however, produce inputs whose phase satisfies $\gamma(\Phi) = \theta \Phi$ for a sign $\theta \in \{\pm 1\}$, for instance Gram matrices. For such inputs, a partner diagonal needs one key switch instead of two. Fix a representative $t \in \mathcal{E}$ with partner t^{-1} ; the partner is computed in three steps.

- **Step 1 (relate the two diagonals).** The conjugate transpose maps diagonal t onto diagonal t^{-1} , $\mu_{t^{-1}} \circ \gamma = g_t^{-1} \circ \mu_t$. Applying $\mu_{t^{-1}}$ to $\gamma(\Phi) = \theta \Phi$ therefore gives

$$\mu_{t^{-1}}(\Phi) = \theta g_{t^{-1}}(\mu_t(\Phi)).$$

- **Step 2 (complete the representative).** The direct route returns e_t with phase $n^{-1}\mu_t(\Phi) - n^{-1}\mu_t(C_{00} + C_{10}s)$, so $e_t + n^{-1}\mu_t(C_{00}, C_{10})$ has phase $n^{-1}\mu_t(\Phi)$.
- **Step 3 (hop).** One Galois step with the linear key $K_{g_{t^{-1}}(s)}$, which PairSwitch already holds, applies $g_{t^{-1}}$ to this phase, and by Step 1

$$e_{t^{-1}} = \theta \cdot \text{Gal}_{g_{t^{-1}}}(e_t + n^{-1}\mu_t(C_{00}, C_{10})) - n^{-1}\mu_{t^{-1}}(C_{00}, C_{10}). \quad (7)$$

Hence this Hermitian hop replaces the two key switches of the merge route by one, and N drops from $2n$ to $\frac{3}{2}n + 1$ with the same keys; Figure 4 illustrates Step 1 on a toy grid of slots. The relation holds between ciphertext phases, requires exact symmetry modulo q , and puts n^{-1} on the hop input.

Inputs that admit the hop. The relation applies to the following inputs.

- **Gram products** $x \otimes x$ and FRee-L sums $\sum_k x_k \otimes x_k$ are symmetric, $\theta = +1$.
- **Symmetrized sums.** Since $\gamma(x \otimes y) = y \otimes x$ for any two ciphertexts, the input $\text{Ct}_4 + \gamma(\text{Ct}_4)$ of a sum $AB^H + BA^H$ is symmetric, which covers symmetric rank- $2k$ updates, the Lyapunov operator $AX + XA^H$ for Hermitian X , and products $M^a M^b$ of one Hermitian M .
- **Commutators.** The input $\text{Ct}_4 - \gamma(\text{Ct}_4)$ of $AB^H - BA^H$ has $\theta = -1$.
- **Squaring chains.** Each level $X_{j+1} = n^{-1}X_j X_j^H$ is a Gram product, which covers powers A^{2^k} , scaling-and-squaring and the Newton–Schulz sign iteration of Hermitian matrices, if a rescale between the Trace product and the switch commutes with γ , as rounding in the basis $\{W^l : 1 \leq l \leq p-1\}$ does.

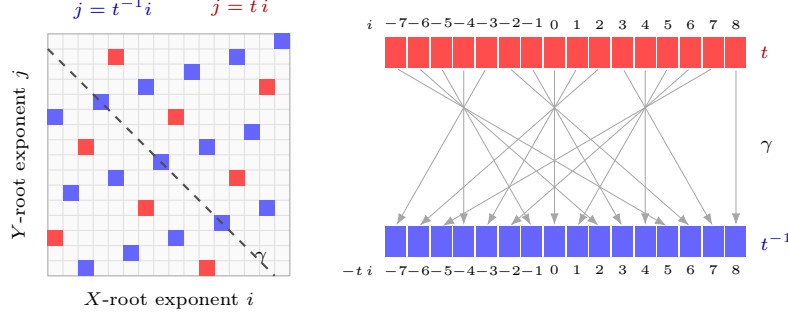


Fig. 4. The Hermitian hop on a toy grid of 16×16 slots with $t = 3$: γ reflects diagonal t onto diagonal t^{-1} across the dashed line, and sends the slot with X -exponent i to the slot with X -exponent $-ti$.

Moreover, applied to the finished output of $x \otimes y$ on all n diagonals, the same relation yields the output of $y \otimes x$, i.e. GL's conjugate transpose of a product, with n base-ring key switches and no second Trace product.

Memory-constrained key sets. Finally, where resident memory rather than upload is the constraint, the pair merge extends to smaller key sets. Split H into babies $B_m = \{5^e : |e| \leq m\}$ and symmetric giants Γ , and write $t = h e$. Then one paired product key $K_{\tau_h^{-1}(s) \kappa \tau_e(s)}$ serves both diagonals of $\{t, t^{-1}\}$, since by the argument of Lemma 2

$$\begin{aligned} x g_t(s) + y s g_t(s) &= \tau_h(\tau_h^{-1}(x) \kappa \tau_e(s) + \tau_h^{-1}(y) \cdot \tau_h^{-1}(s) \kappa \tau_e(s)), \\ x g_{t^{-1}}(s) + y s g_{t^{-1}}(s) &= \psi(\psi^{-1}(x) \tau_h^{-1}(s) + \psi^{-1}(y) \cdot \tau_h^{-1}(s) \kappa \tau_e(s)), \end{aligned}$$

with $\psi = g_{e^{-1}}$. As Figure 5 shows, the resulting family *pwm* spans 243–390 MiB of resident keys at the noise of the split direct route. Together with the baby-step/giant-step sets of Table 9, every key set from 60 MiB on is faster than TFB, and every key set from 229.5 MiB on is at least as fast as GDB-TFB. The paired keys are derivable as well, but not from the layer key alone: each giant $h \in \Gamma$ needs one Galois key of source $\tau_h^{-1}(s)$ at modulus PQP' , so that the upload of pw22 is 150.9 MiB.

Step 1 of the hop rests on the following identity of ring maps, whose proof is given in Appendix B.

Lemma 4 (Hermitian diagonal relation). *For every $t \in H$, $g_t^{-1} \circ \mu_t = \mu_{t^{-1}} \circ \gamma$ as maps $R'_q \rightarrow R_q$. Consequently, if $\gamma(\Phi) = \theta \Phi$ exactly in R'_q for a sign $\theta \in \{\pm 1\}$, then $\mu_{t^{-1}}(\Phi) = \theta g_{t^{-1}}(\mu_t(\Phi))$. Moreover, the phase of every self-product $x \otimes x$, and of every sum of self-products, satisfies $\gamma(\Phi) = \Phi$ exactly.*

5.3 Integration into PairSwitch

Both optimizations plug into the pipeline of Algorithm 3 without changing its structure. On the one hand, the expansion of Algorithm 5 runs once, between key

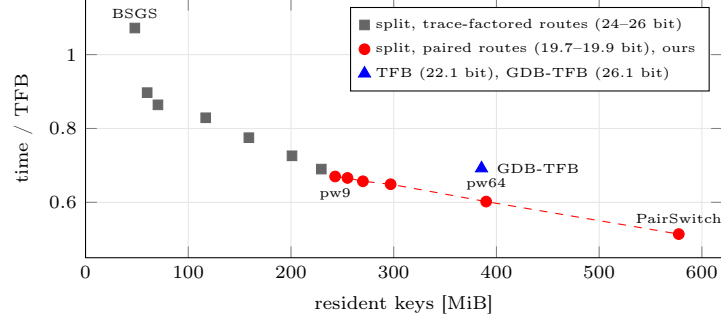


Fig. 5. Resident keys versus BigSwitch time at $n = 256$ (8 threads).

generation and the first BigSwitch, and its output replaces the published product keys. On the other hand, the hop of (7) is a third route for the partner diagonals, which the evaluator selects whenever the caller declares the input symmetric or antisymmetric; likewise, the transposed product is an optional final pass over the finished output. Algorithm 6 states the resulting PairSwitch in full.

Algorithm 6 PairSwitch

KeyGen(s): ▷ client
1: publish $K_{g_t(s)}$ for all $t \in H$ at modulus PQ , and K'_{s_2} at modulus PQP' with $P' = 2^{14} - 1$
Setup: ▷ evaluator, once
2: $(K_{s_{g_t(s)}})_{t \in \mathcal{E}} \leftarrow$ Algorithm 5
BigSwitch(Ct_4, θ, tr), with $\theta \in \{0, +1, -1\}$ and $\gamma(\Phi) = \theta\Phi$ if $\theta \neq 0$: ▷ evaluator, per call
3: **for all** $t \in \mathcal{E}$ **in parallel do** ▷ representatives: direct route
4: $x_t \leftarrow n^{-1}\mu_t(C_{01}), y_t \leftarrow n^{-1}\mu_t(C_{11})$
5: $e_t \leftarrow \text{ModDown}_P(KS^{PQ}(x_t; K_{g_t(s)}) + KS^{PQ}(y_t; K_{s_{g_t(s)}}))$
6: **end for**
7: **for all** partners $t \notin \mathcal{E}$ **in parallel do**
8: **if** $\theta \neq 0$ **then**
9: $e_t \leftarrow \theta \cdot \text{Gal}_{g_t}(e_{t-1} + n^{-1}\mu_{t-1}(C_{00}, C_{10})) - n^{-1}\mu_t(C_{00}, C_{10})$ ▷ hop (7)
10: **else**
11: $e_t \leftarrow$ Algorithm 4 on $(n^{-1}\mu_t(C_{01}), n^{-1}\mu_t(C_{11}))$ ▷ pair merge
12: **end if**
13: **end for**
14: $(E_0, E_1) \leftarrow (C_{00}, C_{10}) + \sum_{t \in H} \delta_t e_t$ ▷ slot scatter $\times n$
15: **if** tr **then** ▷ transposed product $y \otimes x$
16: **for all** $t \in H$ **in parallel do**
17: $f_t \leftarrow \text{Gal}_{g_t}(n^{-1}\mu_{t-1}(E_0, E_1))$ ▷ Lemma 4, $\gamma(\Phi_{x \otimes y}) = \Phi_{y \otimes x}$
18: **end for**
19: **return** (E_0, E_1) and $\sum_{t \in H} \delta_t f_t$
20: **end if**
21: **return** (E_0, E_1)

Theorem 2 (PairSwitch). *For every Ct_4 and every admissible θ , Algorithm 6 returns a BigSwitch output in the sense of Definition 1, whose error is the scattered sum of the per-diagonal errors. It publishes $n + \rho$ base-ring keys, holds $\frac{3}{2}n + 1$ of them after the setup, and makes $2n$ base-ring key switches per call, or $\frac{3}{2}n + 1$ if $\theta \neq 0$, with no Y -axis transform; the transposed product costs n more.*

Proof. Each e_t has phase (2) plus its error: on the representatives by the direct route, on the partners by Proposition 2 or, if $\theta \neq 0$, by Lemma 4, since the hop maps $n^{-1}\mu_{t-1}(\Phi)$ to $\theta n^{-1}\mu_t(\Phi)$ up to one switching error. Summing over t with the scatter gives $C_{01}s_Y + C_{11}s_{s_Y}$ by (5). For the transposed product, Lemma 4 applied to $\Phi_{y \otimes x} = \gamma(\Phi_{x \otimes y})$ gives $\mu_t(\Phi_{y \otimes x}) = g_t(\mu_{t-1}(\Phi_{x \otimes y}))$, and the counts follow from Proposition 3.

5.4 Security

Security sketch. PairSwitch publishes the Galois keys $K_{g_t(s)}$ for $t \in H$ at modulus PQ and the layer key K'_{s^2} at modulus PQP' , all with fresh errors, and every product key is a deterministic public function of these keys by Lemma 3. A reduction that is given this tuple can therefore run Algorithm 5 and the whole evaluator itself, so that GL with PairSwitch is IND-CPA secure as long as RLWE samples under s remain pseudorandom given the tuple, a circular-security assumption on Galois and relinearization keys. This assumption is of the same kind as Assumption 3 of Park et al. [19]: by Proposition 1, their big key K_{s_Y} decomposes into the same Galois keys with correlated errors of standard deviation $\sqrt{n}\sigma$, and a smoothing argument in Appendix C relates the two tuples only up to a factor $c\sqrt{n}$ in the error, with $c \leq 1.44$: Galois keys at error σ yield a secure big key, whereas a secure fresh big key yields secure Galois keys only at error $c\sqrt{n}\sigma$. The two assumptions are thus of the same kind but not equivalent, and ours concerns in addition a layer key at the larger modulus PQP' . Unlike the original BigSwitch, PairSwitch publishes no key of a product source.

6 Implementation and Evaluation

In this section, we first compare one BigSwitch of PairSwitch with the BigSwitches of Gentry and Lee and of Park et al. [19] at identical parameters. We then attribute the gain to the two steps of Section 4, and finally measure whole workloads, including the GPT-2 attention kernel of Park et al. [19]. Our implementation, all experiment scripts and raw logs are public.²

6.1 Experimental Setup

Engine. All schemes run in one C++17 engine (g++-11, OpenMP) with exact RNS/NTT arithmetic over R'_q , separable per-axis transforms and hybrid key

² <https://github.com/acprk/pairswitch-gl>

switching. Specifically, the X and Y axes use Cooley–Tukey and Gentleman–Sande transforms for $X^n = \pm\iota$ and $Y^n = \pm\iota$, and the W axis uses Rader’s algorithm, as in Park et al. [19]. The Trace product computes the four blocks as slot-wise matrix products over $\mathbb{Z}_q$ with an AVX-512 kernel and is coefficientwise equal to an independent exact GL implementation.

Schemes. We compare five BigSwitches, all with the same limbs, digits, key format, transforms and memory layout:

- the Gentry–Lee BigSwitch of Algorithm 1, with one shared ModDown;
- TFB, Algorithm 2, which reproduces the key sizes of Park et al. [19] byte-exactly; we use an optimized implementation whose output is bit-identical to their algorithm, and which is $1.61\times$ and $1.03\times$ faster than a literal one at 8 threads and at one;
- GDB-TFB, i.e. Algorithm 3 with trace-factored routes;
- the split direct route, i.e. Algorithm 3 with the direct route on every diagonal and all n product keys; and
- PairSwitch, Algorithm 6, with every diagonal as one task of an OpenMP loop.

Since the implementation of Park et al. [19] is not public, every comparison runs within one engine, so ratios are fair while absolute times differ.

Parameters. Unless stated otherwise, $n = 256$ and $p = 17$, so $d = 8192$, and we use the chain of Park et al. [19]: three 47-bit Q limbs, a 59-bit P , $\ell = 3$ and $\log_2 PQ = 200$. The input is a real general GL product $x \circledast y$, and the noise is $\log_2$ of the ℓ_2 norm of the output error on two independent inputs. The layer modulus only affects the one-time expansion: the layers $P' = 2^{14} - 1$ of Section 5.1, $2^{15} - 1$ and 87041 give the same key size and noise, and BigSwitch times within 0.4% of each other in paired runs. Tables 1 and 3 use the final layer $P' = 2^{14} - 1$, and the workloads of Section 6.4 use $P' = 87041$.

Host and statistics. We use a shared dual-socket Intel Xeon Gold 6230R host (2×26 cores, 2.1 GHz, AVX-512), and compile with `g++-11 -O3 -march=native`. Every run uses `nice`, at most 8 threads pinned to distinct physical cores of one NUMA node, and all variants interleaved in rotating order in one process; the warm-up repetition is discarded, and we report medians of 7–15 repetitions, 15 for Table 3 at 8 threads, and fewer for the long workloads. Since other jobs share a CPU quota with ours, absolute times drift between runs; the runs of Tables 1 and 3 were taken on a quiet host, at a load average of 4–9; therefore ratios are paired per repetition within one run, and cells from different runs are never divided. Keys are reported unseeded.

Correctness. Every variant decodes correctly in every run, and all workload outputs are exact over $\mathbb{Z}$. As a negative control in every run, a product key expanded from the wrong linear key yields an error of $2^{46} \approx q$ on its diagonals. Moreover, the executed counts of key switches and NTTs match Proposition 3.

6.2 Comparison with Prior Work

Note first that TFB is slower than the Gentry–Lee BigSwitch by design: it trades half of the keys for a third key switch per diagonal. Park et al. [19] report

Table 3. One BigSwitch at $n = 256$, $p = 17$ and $\log_2 PQ = 200$, all variants in one run per thread count. Times are medians in seconds; ratios are paired medians against TFB; noise is $\log_2 \ell_2$ of the output error; keys in MiB.

scheme	keys		8 threads		1 thread		noise
	publ.	resid.	time	ratio	time	ratio	
Gentry–Lee [10]	768.0	768.0	0.929	0.691	6.557	0.667	15.37
TFB [19]	385.5	385.5	1.345	1	9.844	1	22.07
GDB-TFB	385.5	385.5	0.925	0.686	6.874	0.696	26.08
split direct route	394.6	768.0	0.646	0.479	4.712	0.476	19.37
PairSwitch	394.6	577.5	0.680	0.506	4.957	0.501	19.67

2.979s against 4.181s on one core, i.e. +40.4%, and we measure +45% at 8 threads and +50% at one; the absolute times differ because the CPU and the implementation differ, which is why all our comparisons are paired within one engine. As shown in Table 3, PairSwitch takes $0.506\times$ the time of TFB at 8 threads and $0.501\times$ at one, and $0.731\times$ and $0.751\times$ that of the Gentry–Lee BigSwitch, respectively. Four further runs on a more loaded host, with the layers $2^{15}-1$ and 87041, gave $0.502\text{--}0.530\times$ TFB and $0.727\text{--}0.779\times$ Gentry–Lee, so that over all runs PairSwitch is $1.89\text{--}2.00\times$ faster than TFB and $1.28\text{--}1.38\times$ faster than the Gentry–Lee BigSwitch. At the same time, it has 2.4 bits less noise than TFB and publishes 2.4% more bytes, since its 10.6 MiB layer key replaces the 1.5 MiB relinearization key of TFB; in exchange, the evaluator holds $1.5\times$ the keys of TFB after the one-time expansion, which takes 3.3s at 8 threads and 25.7s at one and pays for itself after about five BigSwitch calls. Finally, on the strict two-level layout of Appendix F, where the layer key is the 199-bit master level, PairSwitch takes $0.565\times$ the time of TFB at 8 threads and $0.573\times$ at one.

The split direct route. The split direct route isolates the effect of the pairing. It runs the same $2n$ key switches as PairSwitch with 8192 instead of 8700 NTTs, and if its product keys are derived as in Section 5.1, it publishes the same keys. It is about 5% faster than PairSwitch ($0.951\times$ and $0.948\times$ in paired runs) and has 0.3 bits less noise, but it holds 768 instead of 577.5 MiB of keys and expands twice as many product keys. Hence, on general inputs, the speed of PairSwitch over TFB comes from the splitting and from the product keys, while the pairing trades about 5% of time for 25% of the resident keys, which by Theorem 1 is the least possible at $2n$ key switches. On γ -symmetric inputs, the Hermitian hop needs exactly the keys of PairSwitch and makes $\frac{3}{2}n + 1$ key switches, so that the additional product keys of the split direct route bring no benefit there.

6.3 Sources of the Speedup

The gain of PairSwitch over TFB is the product of the two steps of Section 4; the numbers below are from Table 3.

- **Galois-diagonal pipelines** remove the Y -axis transforms: GDB-TFB, which still runs three switches per diagonal, takes $0.686\times$ and $0.696\times$ the time of

Table 4. Workloads at $n = 256$, $p = 17$ and 8 threads: medians of the Trace products, the BigSwitches and the total, taken separately, and the paired median of the total against TFB. All outputs are exact.

scheme	normal equations, $K = 16$				GPT-2 $O = PV$, $K = 4$			
	Trace	BigS.	total	ratio	Trace	BigS.	total	ratio
TFB [19]	3.24	1.53	4.79	1	5.40	6.15	11.60	1
GDB-TFB	3.28	1.06	4.34	0.904	5.41	4.24	9.70	0.834
PairSwitch	3.20	0.78	4.05	0.836	5.48	3.14	8.66	0.740
PairSwitch + hop	3.24	0.61	3.86	0.812	not a Gram product			

TFB; this matches the removed Y share of the transforms, since a micro-benchmark of mixed versus full transforms gives 0.697.

- **Pair merging** removes one of the three switches on every diagonal: PairSwitch runs 8700 instead of 12288 NTTs, a model ratio of 0.708, and takes $0.736\times$ the time of GDB-TFB at 8 threads and $0.718\times$ at one.

Since a key switch on one core is transform-bound, with the multiply-accumulate at 4–8% of the time and the transforms at 79–86%, the transform count predicts the time; at 8 threads, memory traffic adds a few percent.

6.4 End-to-End Workloads

Table 4 reports two workloads. The first is the normal-equations tile $Z^T Z$ for 255 features and 65536 samples, in BGV with 7-bit data on a 116-bit chain. Here the BigSwitch is a third of the server time, and PairSwitch cuts the total to $0.836\times$ that of TFB, or $0.812\times$ with the Hermitian hop, since $Z^T Z$ is a Gram sum. The second is the GPT-2 attention kernel $O = P \cdot V$ that Park et al. [19] use to motivate TFB, with $L = 1024$ tokens, 12 heads and $K = 4$ accumulated Trace products per output tile. There, the four BigSwitches are 53% of the kernel; PairSwitch halves them and runs the kernel at $0.740\times$ the time of TFB, with all four output tiles exact. Note that our Trace kernel is about $3\times$ faster than the one reported by Park et al. [19], so the BigSwitch share, and hence our gain, is larger here than in their code. For long contexts, by contrast, the shared Trace prefix dominates: at $K = 160$ and $K = 512$ inner tiles, the single BigSwitch per output tile takes 6.0% and 2.1% of the time with TFB and 2.1% and 0.8% with PairSwitch, measured within the same runs on a loaded host.

Structured workloads. Table 5 runs the extensions of Section 5.2 end to end. First, a four-step squaring chain with the hop at every level takes $0.48\times$ the time of the same chain with TFB. The hop alone saves 14% over PairSwitch, which is less than its 20% per BigSwitch because the Trace products and modulus switches are untouched. Second, for shared-query attention scores QQ^T with $L = 1024$ and 4×4 tiles of size 256, the four diagonal tiles are Gram products and the twelve others come in transposed pairs; hence six tiles need neither a Trace product nor a BigSwitch, but only the transpose hops. The layer then takes

Table 5. Structured workloads at $n = 256$, $p = 17$ and 8 threads in BGV; paired medians. The squaring chains extend the limb format to 5 limbs and measure work per level, not a 128-bit parameter set. PS denotes PairSwitch and PS⁺ PairSwitch with the structure-specific steps.

workload	PS/TFB	PS ⁺ /PS	PS ⁺ /TFB
Gram product, one BigSwitch	0.514	0.786	0.408
squaring chain, $k = 4$	0.544	0.859	0.475
shared- QK scores, $L = 1024$	0.443	0.732	0.326
SYR2K $AB^H + BA^H$	–	0.808	–
product pair AB^H, BA^H	–	0.812	–

0.33× the time of TFB on all sixteen tiles, and 0.43× that of a structure-aware TFB that transposes with its R' -wide key.

Large products. Finally, we run two 4096×4096 workloads on the 116-bit chain: the Gram system $X^T X$ and $X^T y$, with $136 + 16$ GL products, and a general product AB , with 256 GL products. Every output entry is exact. Both runs were throttled by the shared CPU quota, which slows the R' -wide TFB more than the R -sized pipelines; hence we report the ratios against GDB-TFB, which runs the same pipeline as PairSwitch. Under this comparison, PairSwitch takes 0.868× the server time of GDB-TFB on the Gram system, 0.812× with the Hermitian hop, and 0.822× on the general product. For completeness, the same runs give 0.380, 0.362 and 0.588 against TFB, inflated by the throttling.

7 Conclusion

We presented PairSwitch, a Galois-paired BigSwitch for Gentry–Lee matrix FHE. After the Galois splitting, the diagonals of a BigSwitch form inverse pairs, so that the two diagonals of a pair share one product key at the cost of one Galois step; the resulting $\frac{3}{2}n + 1$ base-ring keys are optimal among per-diagonal BigSwitches with $2n$ key switches, and all product keys are derived publicly from the Galois keys and one small layer key. The conjugate-transpose symmetry further lowers Gram products to $\frac{3}{2}n + 1$ key switches. On the parameters of Park et al. [19], PairSwitch runs a BigSwitch 1.89–2.00× faster than TFB and 1.28–1.38× faster than the original one, with 2.4 bits less noise than TFB, and it accelerates the GPT-2 attention kernel of Park et al. by 1.35× in our engine.

Two limitations remain. The evaluator holds 1.5× the keys of TFB after the expansion, and the layer key sits at the 128-bit bound of the default cost model of the lattice estimator. Future directions include the other big key switches of GL, such as those of bootstrapping [8], and GPU implementations in which the paired product keys are streamed instead of stored.

References

1. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic encryption standard. In: Lauter, K., Dai, W., Laine, K. (eds.) *Protecting Privacy through Homomorphic Encryption*, pp. 31–62. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77287-1_2
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology* **9**(3), 169–203 (2015). <https://doi.org/10.1515/jmc-2015-0016>
3. Bae, Y., Cheon, J.H., Hanrot, G., Park, J.H., Stehlé, D.: Fast homomorphic linear algebra with BLAS. *Journal of Cryptology* **39**(3), 25 (2026). <https://doi.org/10.1007/s00145-026-09580-x>
4. Bossuat, J.P., Mouchet, C., Troncoso-Pastoriza, J., Hubaux, J.P.: Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In: *Advances in Cryptology – EUROCRYPT 2021*. LNCS, vol. 12696, pp. 587–617. Springer (2021). https://doi.org/10.1007/978-3-030-77870-5_21
5. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: *ITCS 2012*. pp. 309–325 (2012)
6. Cheon, J.H., Kang, M., Park, J.H.: Towards lightweight CKKS: On client cost efficiency. In: *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIACCS ’26)*. pp. 1387–1398. ACM (2026). <https://doi.org/10.1145/3779208.3785268>
7. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: *ASIACRYPT 2017*. LNCS, vol. 10624, pp. 409–437 (2017)
8. Crockett, E., Gentry, C., Kim, H., Lee, Y., Lee, Y.: Efficient bootstrapping in fully homomorphic encryption for matrix arithmetic. In: *Advances in Cryptology – CRYPTO 2026*. LNCS, vol. 16801, pp. 492–524. Springer (2026). https://doi.org/10.1007/978-3-032-35374-0_16
9. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: *Advances in Cryptology – CRYPTO 2012*. LNCS, vol. 7417, pp. 850–867. Springer (2012). https://doi.org/10.1007/978-3-642-32009-5_49
10. Gentry, C., Lee, Y.: Fully homomorphic encryption for matrix arithmetic. In: *Advances in Cryptology – CRYPTO 2026*. LNCS, vol. 16801, pp. 363–395. Springer (2026). https://doi.org/10.1007/978-3-032-35374-0_12
11. Halevi, S., Shoup, V.: Faster homomorphic linear transformations in HElib. In: *Advances in Cryptology – CRYPTO 2018*. LNCS, vol. 10991, pp. 93–120. Springer (2018). https://doi.org/10.1007/978-3-319-96884-1_4
12. Han, K., Ki, D.: Better bootstrapping for approximate homomorphic encryption. In: *Topics in Cryptology – CT-RSA 2020*. LNCS, vol. 12006, pp. 364–390. Springer (2020). https://doi.org/10.1007/978-3-030-40186-3_16
13. Jiang, X., Kim, M., Lauter, K., Song, Y.: Secure outsourced matrix computation and application to neural networks. In: *ACM CCS 2018*. pp. 1209–1222. ACM (2018). <https://doi.org/10.1145/3243734.3243837>
14. Kim, A., Lee, Y., Deryabin, M., Eom, J., Choi, R.: LFHE: Fully homomorphic encryption with bootstrapping key size less than a megabyte. *Cryptology ePrint Archive*, Paper 2023/767 (2023), <https://eprint.iacr.org/2023/767>

15. Lattice estimator. <https://github.com/malb/lattice-estimator>, software repository; accessed 2026-09-10
16. Lee, J.W., Lee, E., Kim, Y.S., No, J.S.: Rotation key reduction for client-server systems of deep neural network on fully homomorphic encryption. In: Advances in Cryptology – ASIACRYPT 2023. LNCS, vol. 14443, pp. 36–68. Springer (2023). https://doi.org/10.1007/978-981-99-8736-8_2
17. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Journal of the ACM **60**(6), 43:1–43:35 (2013)
18. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. SIAM J. Comput. **37**(1), 267–302 (2007)
19. Park, D.J., Jeong, H., Jeong, M., Oh, J., Lee, Y., Kim, Y.S.: Trace-factored BigSwitch for matrix-friendly FHE. Cryptology ePrint Archive, Paper 2026/2058 (2026), <https://eprint.iacr.org/2026/2058>
20. Park, J.H.: Ciphertext-ciphertext matrix multiplication: Fast for large matrices. In: Advances in Cryptology – EUROCRYPT 2025. LNCS, vol. 15608, pp. 153–180. Springer (2025). https://doi.org/10.1007/978-3-031-91101-9_6
21. Peikert, C.: An efficient and parallel Gaussian sampler for lattices. In: CRYPTO 2010. LNCS, vol. 6223, pp. 80–97 (2010)
22. Zheng, X., Li, H., Wang, D.: A new framework for fast homomorphic matrix multiplication. Designs, Codes and Cryptography **93**(7), 2671–2693 (2025). <https://doi.org/10.1007/s10623-025-01614-y>

A Proof of Lemma 1

Throughout, every prime factor of q is $\equiv 1 \pmod{4np}$, $t, t' \in H$, $g_t = \kappa\tau_t$, and the maps μ_t and elements δ_t are those of Section 3.

Step 1 (Orthogonality). We first evaluate $\mu_{t'}$ on δ_t . Writing $r = X^{t'-t}$,

$$\mu_{t'}(\delta_t) = \sum_{m < n} X^{-mt} X^{mt'} = \sum_{m < n} r^m, \quad r^n = i^{t'-t} = 1,$$

since $t' - t \equiv 0 \pmod{4}$. If $t \neq t'$, then modulo each prime of q the X -slots are primitive $4n$ -th roots of unity ζ with $\zeta^{t'-t} \neq 1$, so $r - 1$ is a unit and the geometric sum vanishes; if $t = t'$, the sum equals n . This proves (a).

Step 2 (Isomorphism). Next, μ is an R_q -algebra map between free R_q -modules of rank n , and by (a)

$$\mu\left(n^{-1} \sum_t \delta_t x_t\right) = (x_t)_{t \in H} \quad \text{for all } (x_t)_t.$$

Hence μ is surjective, thus bijective with the stated inverse, proving (b).

Step 3 (Action on the secret). Finally, μ_t fixes i , X and W . For $r \in R$ we have $\gamma(r) = \bar{r}(Y^{-1}, W^{-1})$ with $i \mapsto -i$, hence

$$\mu_t(\gamma(r)) = \bar{r}(X^{-t}, W^{-1}) = \kappa\tau_t(r).$$

Taking $r = s$ gives $\mu_t(s_Y) = \kappa\tau_t(s)$, and multiplicativity gives $\mu_t(s s_Y) = s \cdot \kappa\tau_t(s)$, which proves (c). $\square$

B Proof of Lemma 4

Step 1 (The identity of maps). Both $g_t^{-1} \circ \mu_t$ and $\mu_{t^{-1}} \circ \gamma$ are ring maps that act on A_q by $i \mapsto -i$ and $W \mapsto W^{-1}$, so it suffices to compare them on X and Y . Since $g_t^{-1} = \kappa \tau_{t^{-1}}$,

$$\begin{aligned} g_t^{-1} \circ \mu_t : X &\mapsto X \mapsto X^{-t^{-1}}, & Y &\mapsto X^t \mapsto X^{-1}, \\ \mu_{t^{-1}} \circ \gamma : X &\mapsto Y^{-1} \mapsto X^{-t^{-1}}, & Y &\mapsto X^{-1} \mapsto X^{-1}, \end{aligned}$$

so the two maps agree.

Step 2 (Symmetric phases). Next, if $\gamma(\Phi) = \theta\Phi$, applying $\mu_{t^{-1}}$ and the first identity gives

$$\theta \mu_{t^{-1}}(\Phi) = \mu_{t^{-1}}(\gamma(\Phi)) = g_t^{-1}(\mu_t(\Phi)) = g_{t^{-1}}(\mu_t(\Phi)),$$

and multiplying by $\theta = \theta^{-1}$ yields the claim.

Step 3 (Self-products). Finally, for the second statement, γ is an involutive ring automorphism with $\gamma(s) = s_Y$, and the Trace product satisfies $\gamma(x \otimes y) = y \otimes x$. Hence a self-product $x \otimes x$ has $\gamma(\Phi) = \Phi$ exactly, noise included. $\square$

C Re-randomizing a Converted Key

The following lemma relates the Galois keys of Proposition 1 to a fresh big key; here $D = nd$ is the dimension of a key over R' , η_ε is the smoothing parameter, and $D_{A,\sigma}$ denotes the discrete Gaussian of standard deviation σ .

Lemma 5 (re-randomizing a linear image of a key). *Let $M \in \mathbb{Z}^{D \times D}$ be invertible modulo q , $s_0 = \sqrt{2\pi}\sigma > \eta := \eta_\varepsilon(\mathbb{Z}^D)$, $r' = (\eta^{-2} - s_0^{-2})^{-1/2}$ and $c = \sqrt{1 + r'^2/s_0^2}$. If $e \leftarrow D_{\mathbb{Z}^D, \sigma}$ and $f \leftarrow D_{\mathbb{Z}^D, \rho}$ with $\rho^2 = c^2\|M\|^2\sigma^2 - \sigma^2MM^\top$, then $Me + f$ is within statistical distance 8ε of $D_{\mathbb{Z}^D, c\|M\|\sigma}$. For continuous Gaussians the same holds with $c = 1$ and distance 0.*

Proof. Step 1 (Setup). We have $Me \sim D_{\Lambda_2, \sqrt{\Sigma_2}}$ with $\Lambda_2 = M\mathbb{Z}^D \subseteq \mathbb{Z}^D$ and $\Sigma_2 = s_0^2MM^\top$. Put $r = c\|M\|s_0$ and $\Sigma_1 = r^2I - \Sigma_2$, and apply the convolution theorem of Peikert [21, Thm. 3.1] with $\Lambda_1 = \mathbb{Z}^D$.

Step 2 (First condition). Since $\|M\| \geq 1$ and $r' \geq \eta$,

$$\Sigma_1 \succeq (r^2 - s_0^2\|M\|^2)I = \|M\|^2r'^2I \succeq \eta^2I.$$

Step 3 (Second condition). With $\tau = (r^2 - s_0^2\|M\|^2)^{-1}$ we have $\Sigma_1^{-1} \preceq \tau I \preceq \tau s_0^2\|M\|^2 \Sigma_2^{-1}$, hence

$$\Sigma_3 := (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1} \succeq \frac{\Sigma_2}{1 + \tau s_0^2\|M\|^2} = \frac{\eta^2}{s_0^2} \Sigma_2 = \eta^2 MM^\top.$$

As $(MM^\top)^{-1/2}M$ is orthogonal and smoothing is monotone, $\eta_\varepsilon(\Sigma_3^{-1/2}A_2) \leq 1$.

Step 4 (Continuous case). Finally, $Me+f$ is Gaussian with covariance $\sigma^2 MM^\top + \rho^2 = c^2 \|M\|^2 \sigma^2 I$. $\square$

Using $\eta_\varepsilon(\mathbb{Z}^D) \leq \sqrt{\ln(2D(1+1/\varepsilon))/\pi}$ [18, Lemma 3.3] at $\sigma = 3.2$, we obtain $c = 1.191$ for $\varepsilon = 2^{-64}$ and $c = 1.435$ for $\varepsilon = 2^{-128}$ at $D = 2^{21}$. For the map M of Proposition 1 we have $MM^\top = nI$, so $\|M\| = \sqrt{n}$. Consequently, since pseudorandomness is preserved by public invertible maps, Galois keys at noise σ that are pseudorandom yield a pseudorandom big key, whereas a pseudorandom fresh big key yields pseudorandom Galois keys only at noise $c\sqrt{n}\sigma$.

D Proof of Theorem 1

D.1 The Per-Diagonal Model

Fix a diagonal t and write $g = g_t$. Noise is ignored, and correctness is required as an identity for all $s \in R_q$, all public inputs and all key masks, which every construction of this paper satisfies; this is stronger than correctness for ternary s only, so the bound below applies to fewer programs.

Definition 2 (per-diagonal model). *A route for t is a straight-line program with public inputs $x, y \in R_q$. Its values are public elements of R_q , computed from x, y and public constants, and ciphertexts $(c_0, c_1) \in R_q^2$ with phase $c_0 + c_1 s$. It uses three kinds of steps:*

- *key applications $\text{KS}(z; K)$, whose input $z \in R_q$ is any value, including an automorphic image of a component of an earlier output, and whose output is $(z\lambda_K(s) - a s, a)$ with a fresh uniform mask a ; the source λ_K is $h(s)$ for an automorphism $h \neq \text{id}$, or $h_1(s)h_2(s)$;*
- *key-free ciphertexts (d_0, d_1) with public d_0, d_1 ;*
- *sums of ciphertexts and of their automorphic images with coefficients in $\mathbb{Z}$.*

The route is correct if its output has phase $xg(s) + yg(s)$. Its cost N_t is the number of key applications, and a per-diagonal BigSwitch runs one route per diagonal and holds the keys used by all of them.

The integer coefficients reflect that a ciphertext can be multiplied by a large public value only through a key application, whose gadget decomposition keeps the noise small. The two slot-level facts that we use are the following.

- *Monomials.* Since the Galois group acts freely on the slots, the slot values of s and of its automorphic images are distinct coordinates, which are independent over $\mathbb{Z}_q$ by the CRT. Hence distinct monomials in s , its automorphic images and the masks are linearly independent over the functions of the public inputs, and we may compare their coefficients.
- *Masks.* An automorphic image $h(a)$ of a mask is a fresh coordinate, so in a correct route its total coefficient in the output phase vanishes.

D.2 Proof

Step 1 (Unused images). Suppose that no later key application takes an image of the output (a_0, a) of a key application A as input. If the output uses $h(a_0)$ and $h(a)$ with integer coefficients $\epsilon, \epsilon', \zeta, \zeta'$ in its two components, the coefficient of $h(a)$ is $-\epsilon h(s) - \epsilon' s h(s) + \zeta + \zeta' s$. For $h \neq \text{id}$ these four monomials are distinct, so all coefficients vanish; for $h = \text{id}$ we get $\epsilon' = \zeta = 0$ and $\zeta' = \epsilon$, i.e. A enters as the ciphertext $\epsilon(a_0, a)$ with phase $\epsilon z \lambda(s)$.

Step 2 (One key application). Then the output phase is $\epsilon z_1 \lambda_1 + d_0 + d_1 s$. The monomials $g(s)$ and $s g(s)$ both carry free inputs, while λ_1 is a single monomial; hence $N_t = 1$ is impossible, and so is $N_t = 0$. Therefore $N_t \geq 2$ for every diagonal, and $N \geq 2n$.

Step 3 (Two key applications, public inputs). Let $A_1 = \text{KS}(z_1; K_1)$ with public z_1 , and $A_2 = \text{KS}(z_2; K_2)$. If z_2 is public as well, the phase is $\epsilon_1 z_1 \lambda_1 + \epsilon_2 z_2 \lambda_2 + d_0 + d_1 s$, and comparing coefficients gives $\{\lambda_1, \lambda_2\} = \{g(s), s g(s)\}$: the direct route.

Step 4 (Two key applications, a reused output). Otherwise $z_2 = w + \sum_h (\rho_h h(a_0) + \sigma_h h(a))$ for the output (a_0, a) of A_1 . The coefficient of $h(a)$ in the output phase is now

$$-\epsilon_h h(s) - \epsilon'_h s h(s) + \zeta_h + \zeta'_h s + \epsilon_2 \lambda_2 (\sigma_h - \rho_h h(s)) = 0.$$

Since $\lambda_2 \neq 1, s$, the term $\rho_h h(s) \lambda_2$ matches none of the others, so $\rho_h = 0$; and if $\sigma_h \neq 0$, then $\lambda_2 \in \{h(s), s h(s)\}$. If $\sigma_h = 0$ for all h , then z_2 is public and Step 3 applies. Otherwise A_2 is a Galois step ($\lambda_2 = h(s)$, $h \neq \text{id}$, $\epsilon_h \neq 0$) or a multiplication by s ($\lambda_2 = s h(s)$, $\epsilon'_h \neq 0$), and the output phase is

$$\begin{aligned} & \epsilon z_1 \lambda_1 + \epsilon_h h(z_1) h(\lambda_1) + \epsilon_2 w h(s) + d_0 + d_1 s && \text{(Galois step), or} \\ & \epsilon z_1 \lambda_1 + \epsilon'_h s h(z_1) h(\lambda_1) + \epsilon_2 w s h(s) + d_0 + d_1 s && \text{(multiplication by } s\text{).} \end{aligned}$$

Step 5 (The Galois step). In the first case, λ_1 and $h(\lambda_1)$ have the same degree. If both were linear, no term could produce $s g(s)$; hence they are quadratic, and the linear monomial $g(s)$, whose coefficient is the free input x , can only come from $w h(s)$, which forces $h = g$. The quadratic monomial $s g(s)$ then comes from λ_1 or from $g(\lambda_1)$. If $\lambda_1 = s g(s)$ and $g^2 \neq \text{id}$, the monomial $g(\lambda_1)$ is not in the target, so $\epsilon_h = 0$ and z_2 is public, which is the previous case. Otherwise $g(\lambda_1) = s g(s)$, i.e. $\lambda_1 = s g^{-1}(s) = s g_{t-1}(s)$: the merge route of Algorithm 4.

Step 6 (The multiplication by s). In the second case, $g(s)$ can only come from $\lambda_1 = g(s)$ with $\epsilon z_1 = x$. Then $s g(s)$ comes either from $s h(\lambda_1)$, which forces $h = \text{id}$ and the coefficient $\epsilon'_h z_1 = \epsilon'_h x / \epsilon$, not the free input y ; or from $w s h(s)$ with $h = g$, in which case $s g^2(s)$ is not in the target, so $\epsilon'_h = 0$ and z_2 is public. Both contradict the assumption or reduce to the direct route.

Step 7 (Counting). Hence a BigSwitch with $N = 2n$ makes exactly two applications per diagonal and uses, on diagonal t , the linear key of source $g_t(s)$ and a product key of source $s g_t(s)$ or $s g_{t-1}(s)$. The n linear sources are distinct. A product source $s g_u(s)$ serves only $t \in \{u, u^{-1}\}$, and inversion on H has $n/2 + 1$

orbits, the fixed points being 1 and $5^{n/2}$; hence at least $n/2 + 1$ product keys are needed. After the setup, once the layer key is deleted, PairSwitch holds exactly these keys. $\square$

For monomial routes, the per-diagonal costs of this model were checked against a shortest-path search over all programs in 2760 random (key set, diagonal) cases at $n = 4, 8$; with these costs, an exhaustive search over all 2^{31} key sets at $n = 8$ gives the frontier quoted in Section 4.4.

E The Phase-Linear Bound

Throughout, q is prime with $q \equiv 1 \pmod{4np}$, noise is ignored, and the secret ranges over a set $\mathcal{S}$ that contains the full ternary grid of Y -free secrets.

E.1 The Phase-Linear Model

Definition 3 (phase-linear key-switching model $\mathcal{M}_{\text{PL}}$). *A BigSwitch is a straight-line program on ciphertexts (c_0, c_1) with phase $c_0 + c_1 s$. A resident key K has rows (u, v) with $u + vs = Pg \lambda_K(s) + e$ for a fixed source map $\lambda_K : \mathcal{S} \rightarrow R'_q$. The program uses three kinds of steps:*

- *public-input key applications $\text{KS}(x; K)$, with x a function of Ct_4 ;*
- *R' -linear combinations with coefficients that depend on Ct_4 , public constants and key masks, and the key-free automorphisms $Y \mapsto Y^t$; an automorphism that moves the X -slots occurs only inside a key application, which is then counted as a runtime step;*
- *runtime key applications, whose input depends on an earlier key application.*

U is the number of resident base-ring keys, and $W = W_0 + W_C$ is the number of runtime steps weighted by input width (1 for R -sized and n for R' -wide inputs), where W_0 counts the steps whose input is computed without Ct_4 (key derivation) together with the applications of derived keys to public inputs, and W_C counts all others. Correctness is required exactly for all $s \in \mathcal{S}$.

Theorem 3 (resident plus runtime work). *Every BigSwitch in $\mathcal{M}_{\text{PL}}$ satisfies $U + W_0 + 2W_C \geq 2n$. If only one of the two terms $C_{01}s_Y, C_{11}s_Y$ is required, it satisfies $U + W \geq n$. Both bounds are tight at $W = 0$.*

Lemma 6 (independence). *Let $q > 2$. Fix a branch ι , an X -root a and a W -root w , and for each Y -root b let $\pi(b)$ be the R -slot with branch $-\iota$, X -root b^{-1} and W -root w^{-1} . Then the $2n + 2$ functions*

$$1, \quad s(a), \quad s(\pi(b)), \quad s(a)s(\pi(b)) \quad (b \text{ a } Y\text{-root})$$

are linearly independent over $\mathbb{Z}_q$ as functions on $\mathcal{S}$.

E.2 Proof of Lemma 6

Step 1 (Distinct slots). Evaluating γ in slots gives $s_Y(a, b) = s(\pi(b))$. The n slots $\pi(b)$ are pairwise distinct and differ from (ι, a, w) , because $(b^{-1})^n = \iota^{-1} = -\iota$.

Step 2 (Change of variables). By the CRT isomorphism, distinct slots give linearly independent $\mathbb{Z}_q$ -linear forms in the d coordinates of s . After an invertible linear change of variables, these $n+1$ forms become coordinates $y_0, \dots, y_n$, and the functions become the distinct monomials $1, y_0, y_b$ and $y_0 y_b$.

Step 3 (Independence on the grid). Finally, a nonzero polynomial of degree at most 2 in each variable cannot vanish on $\{-1, 0, 1\}^d$ over $\mathbb{Z}_q$ with $q > 2$, by induction on the variables; composing with an invertible linear map preserves this. $\square$

E.3 Proof of Theorem 3

Step 1 (Available functions). Fix a branch ι , an X -slot a and a W -slot w . Let $\mathcal{R}_a$ be the span of $1, s(a)$, the values at X -slot a of all resident sources, and those of the outputs of the steps counted in W_0 ; since these do not depend on the input C ,

$$\dim \mathcal{R}_a \leq 2 + U + W_0.$$

For an input C , let $\mathcal{Q}_a(C)$ be the span of the values at X -slot a of the outputs of the steps counted in W_C , so that $\dim \mathcal{Q}_a(C) \leq W_C$.

Step 2 (Invariant). By induction over the program, every phase produced on input C has all its values at X -slot a in $\mathcal{R}_a + \mathcal{Q}_a(C)$: public-input applications contribute $x(a, b)\lambda_K(s)(a, b)$, linear operations act slot-wise, Y -automorphisms permute b within the X -slot a , and runtime outputs are generators.

Step 3 (Two inputs). Let $1_{a,*}$ be the indicator of the n slots (a, b) , and put

$$\Lambda = \langle s(\pi(b)) \rangle_b, \quad \Pi = \langle s(a) s(\pi(b)) \rangle_b, \quad Z = \langle 1, s(a) \rangle \subseteq \mathcal{R}_a.$$

By Lemma 6, $Z + \Lambda + \Pi$ is direct and $\dim \Lambda = \dim \Pi = n$. Correctness on $C_{01} = 1_{a,*}$, with all other blocks zero, gives $\Lambda \subseteq \mathcal{R}_a + \mathcal{Q}_a(C)$, and similarly on $C_{11} = 1_{a,*}$; therefore

$$W_C \geq n - \dim(\Lambda \cap \mathcal{R}_a) \quad \text{and} \quad W_C \geq n - \dim(\Pi \cap \mathcal{R}_a).$$

Step 4 (Conclusion). Since $Z, \Lambda \cap \mathcal{R}_a$ and $\Pi \cap \mathcal{R}_a$ are in direct sum inside $\mathcal{R}_a$,

$$2 + \dim(\Lambda \cap \mathcal{R}_a) + \dim(\Pi \cap \mathcal{R}_a) \leq 2 + U + W_0,$$

and adding the two bounds on W_C gives $2W_C \geq 2n - U - W_0$. For one term, a single input suffices, which gives $W_C \geq n - U - W_0$. $\square$

The proof evaluates the program on two fixed inputs and never compares runtime outputs across inputs; this is why it remains valid when runtime outputs depend on the input, as in Remark 1. The exact ranks of Lemma 6 were checked for $n \leq 8$, and the counts of pwm by a shortest-path search over per-diagonal programs for $n \leq 32$.

F Parameters, Noise and Additional Experiments

Throughout, errors are discrete Gaussians with $\sigma = 3.2$, secrets are uniform ternary, and $(n, p) = (256, 17)$, so $d = 8192$.

F.1 Parameters and Concrete Security

Table 6 lists the modulus chains. The two-level layout uses the inner chain q_0q_1 with $P = q_2$ and four split 24-bit digits, and its 199-bit master level serves as the layer key of the expansion.

Table 6. Modulus chains.

chain	Q limbs (bits)	P (bits)	$\log_2 PQ$	$\log_2 PQP'$
Park et al. [19]	3×47	59	200	214
two-level	2×47	47	141	199
workload	3×30	26	116	132.4

The BGV normal-equations workload uses the plaintext prime $t = 1073655809 \approx 2^{30}$, and the GPT-2 kernel a 36-bit plaintext prime with $t > 2 \max |O|$, so every output is exact over $\mathbb{Z}$. For $d = 8192$, the HE-standard bound for 128 bits is $\log_2 PQ \leq 218$ [1]. Since the GL ring is not a power-of-two cyclotomic, we rely on the lattice estimator [2, 15] instead, for ternary secrets and $\sigma = 3.2$; Table 7 lists its estimates. The largest modulus of PairSwitch is the layer key, and since the three Q primes are below 2^{47} and P below 2^{59} , we have $\log_2(PQP') < 200 + \log_2(2^{14} - 1) < 214$. The workload runs of Section 6.4 used the prime layer $P' = 87041$, i.e. a layer key of 216.4–218.2 bits; the layer affects only the one-time expansion, not the BigSwitch.

Table 7. Lattice-estimator security at $d = 8192$ (bits).

modulus	$\log_2$	default (MATZOV)	classical core-SVP
chain PQ of Park et al. [19]	200	137.7	107.3
layer key PQP' , $P' = 2^{14} - 1$ (ours)	214	128.3	97.2
layer key PQP' , $P' = 2^{15} - 1$	215	127.5	96.7

The estimates for the chain reproduce those of Park et al. [19]. Hence the layer key is at the 128-bit bound of the default cost model, and, like the chain itself, below it under the more conservative core-SVP model; a deployment that requires core-SVP security must shorten the chain for every scheme in our tables, and the layer then follows the same rule. The two-level layout needs no extra layer, since the 199-bit master level of its chain serves as the layer key. The prototype uses a non-cryptographic PRG; it is a measurement tool.

F.2 Noise Analysis

We use the standard heuristic that distinct automorphic images and distinct digits contribute independent errors. For one R -sized switch, let v_k and v_r be the per-coefficient variances of the key term and of the rounding term. Since the coefficient of each monomial $X^j Y^m$ in $\delta_t \varepsilon_t$ is one signed coefficient of ε_t , every output coefficient receives one term from each diagonal, as in Table 8.

Table 8. Output variance per coefficient and per switched term.

BigSwitch	variance
R' -wide switch (Gentry–Lee)	$n v_k + v_r$
split, one ModDown per diagonal (GDB-TFB, PairSwitch)	$n v_k + n v_r$
split, scatter at PQ , one R' -wide ModDown	$n v_k + v_r$
TFB, with v_V the rounding variance of V	$n v_k + v_r + \ s\ ^2 v_V$

Moreover, a Galois step permutes its input error, so the merge route has the variance of two switches on its diagonal, and the Hermitian hop that of its representative plus one switch. For derived product keys, the row error $s e + e'$ of about $2^{8.4}$ per coefficient enters a BigSwitch as $P^{-1} \sum_h D_h e'_h \approx 2^{1.9}$, where D_h are the digits of the switched input, below the rounding term of about $2^{4.4}$; this predicts at most +0.05 bit, and we measure +0.01 bit. In the engine, the split form costs +4.00 bits at $n = 256$.

F.3 Additional Experiments

Correctness checks. Every variant decodes correctly in every run. In addition, a product key expanded from the wrong linear key gives an error of $2^{46} \approx q_0$ on its diagonals, a wrong-orientation paired key and a key derived with swapped layer keys give garbage as well, and the executed counts of key switches and NTTs equal the plan. With the hop disabled, the output equals the split BigSwitch.

All key sets. Table 9 lists the key sets behind Figure 5, all from one run, which is a different run from Table 3. The upload column is computed, not measured: it assumes that the product keys are derived with the layer key as in Section 5.1, and that the paired keys of *pwm* are derived with one further layer Galois key per giant as in Section 5.2, whose number decreases with m ; hence the upload is not monotone in the resident size. The noise column is measured with fresh keys; deriving the paired keys of *pwm* adds 0.28–0.41 bits, and deriving the product keys of PairSwitch adds 0.01 bits. As for the key sets, BSGS stores baby-step and giant-step Galois keys and relinearizes at run time, and P-giant stores product keys on the giant steps and reaches every diagonal by a baby step; the rows between them are hybrids that add a few product keys or merged diagonal pairs to these two sets, and *pwm* is the paired family of Section 5.2.

The two-level layout. On the two-level chain of Table 6, the upload is 385.5 MiB against 384.75 MiB for TFB, and PairSwitch takes $0.565 \times$ the time of TFB at 8 threads and $0.573 \times$ at one thread, with 9085 NTTs against 11008 and 19.66 bits of noise; the derived product keys add no measurable noise there.

Table 9. Resident keys (MiB), upload (MiB), key switches with runtime switches in parentheses, noise ($\log_2 \ell_2$), and paired time ratios at 8 threads.

key set	resident	upload	N	noise	/GDB-TFB	/TFB
BSGS	48.0	48.0	1233 (736)	26.34	1.577	1.072
BSGS, 9 product keys	60.0	57.1	991 (592)	25.58	1.321	0.897
P-giant	70.5	58.6	1008 (496)	25.63	1.302	0.884
16 babies, products	70.5	57.1	977 (480)	25.54	1.269	0.864
P-giant, 32 pairs	117.0	104.1	941 (429)	25.45	1.199	0.829
P-giant, 64 pairs	159.0	146.6	877 (365)	25.19	1.120	0.775
P-giant, 96 pairs	201.0	189.1	813 (301)	24.77	1.055	0.726
P-giant, 119 pairs	229.5	210.4	768 (256)	24.21	0.997	0.690
pw9	243.0	208.9	749 (246)	19.86	0.972	0.670
pw16	255.0	157.1	735 (239)	19.85	0.971	0.666
pw22	270.0	150.9	723 (233)	19.84	0.951	0.657
pw32	297.0	156.6	703 (223)	19.83	0.938	0.649
pw64	390.0	228.4	639 (191)	19.78	0.870	0.602
PairSwitch	577.5	394.6	512 (127)	19.67	0.738	0.514
GDB-TFB	385.5	385.5	768 (256)	26.08	1	0.692
TFB	385.5	385.5	768 (256)	22.08	1.445	1