DVA: Dangerous Variations of ALTEQ

Arnaud Sipasseuth[0000−0003−1048−4822]

KDDI Research Inc, Japan
xan-shipasata@kddi.com

Abstract. In this paper, we present three types of variations of the ALTEQ cryptosystem, a recent submission to the NIST's additional call for signatures. We name these Dangerous Variations of ALTEQ (DVA), as there is always a certain danger in stepping out of usual constructions, although we attempt to maintain heuristic security. First, we present DVA-GG (Graph Generalization), that can be seen as a more abstract point-of-view on the operations done in ALTEQ and encourages more research on the algebraic variants. In particular, we show this approach can lead to a patch counter to Beullens' recent seed collision attack on ALTEQ that only depends on the primitive, and showcase some fancy usages of the primitive for experimental protocols. Second, we present DVA-PC (Precomputations) which is "likely" as secure as ALTEQ in the random oracle model, and allow to drastically reduce the intermediate memory requirements within both the signature and verification process through an easily parallelizable extra operation. In particular, this facilitates precomputation variants with online phases that only depends on the complexity of basic matrix operations. We can then choose between either a tiny offline memory per signature, or get one of the fastest online signing speed for post-quantum cryptography. Third, we present DVA-DM (Distinct Matrices), some cryptanalytic targets that deviates from ALTEQ's original algebraic structure. Those structures can serve as plain computational acceleration or just compress data sizes, and provide good options to motivate the study of specialized cryptanalysis for ALTEQ: if those are safe, then ALTEQ gain safe variants, and otherwise, we gain further understanding of the problems. In particular, the ideas can be applied beyond ALTEQ and beyond, and hopefully extend to MEDS, LESS, and group-action-based cryptography.

Keywords: Post-Quantum Cryptography · Signature scheme · Alternate Trilinear Forms

1 Introduction

Many cryptographers believe that we need to prepare against a powerful tool that would shake the foundations of our world: quantum computers. Indeed, most tools that we use today to certify the validity of our communications and to protect our privacy, from a state level to an individual level, would be broken if such a machine come to life. The reason is simple: the underlying mathematical problems we use to ensure the difficulty of breaking our codes are simple to solve in a quantum world. The most famous example is how RSA-based algorithms [39] would be broken by Shor's algorithm [42].

Thus, the NIST launched a standardization project [33] to spread post-quantum cryptography, in which they asked the academic community to submit cryptosystems that would be resilient against quantum computers. After several years of debates and some studies, a few candidates were chosen by the NIST to be standardized [34], but it does seem that those selected were not enough. An additional call for signatures [36] was made, attracting new primitives as submissions for post-quantum cryptography. This call is looking to expand over the already selected signature schemes, and while we have no clue what are the exact criteria used by the NIST, it is at least written in black and white on their webpage that they aim to diversify their portfolio and at least consistently said that they *may* be interested in schemes with short signature and fast verification. The statement about lattices [34] make us believe that novel primitives were also of interest[1]. Although academic research should not be limited to following NIST requests, the efforts pushed by the NIST have the merit to welcome research directions that have not been seriously considered before in the academic community, such as implementation-focused directions, anything that lack a "security proof", or have but judged practically inefficient, etc[2]. The `[pqc-forum]` now allows to have healthier feedback loops between attacks, patches, testing of practical parameters, and so on[3]. Implementations and practical parameters, considerations, have seemingly gained value as contributions due to the NIST's efforts.

This work on ALTEQ is in somewhat in the same vein, focused on practical considerations and theoretical variations that could bring practical advantages. ALTEQ is a fork of a new post-quantum cryptosystem that saw the light at EUROCRYPT2022 [45], based on the Alternate Trilinear Form Equivalence (ATFE) problem which is believed to be post-quantum. The scheme was cryptanalysed in CRYPTO2023 [10], but the attack was still of an exponential magnitude, thus asymptotically the primitive is clearly still valid. Thus, ALTEQ [13] was submitted to the NIST, as a signature scheme relying on the (ATFE) problem, constructed via a Fiat-Shamir (FS) transform [21] over a Goldreich-Micali-Wigderson (GMW) framework [23] and submitted to the aforementioned additional call for signatures. Less than two days after the submission, a forgery attack found by Saarinen [40] was published followed shortly after with a quick fix suggested by Beullens [8] (who also provided further analysis, affecting somehow similar schemes such as [6] and [17]). The fix was implemented, and an updated

[1] Since we never directly interacted with the NIST, take *any* assumption or supposition we make with a grain of salt: the only sources available to are is their website, random rumours and second-hand information.

[2] New drawback however: non-NIST related works may be prematurely shot down.

[3] In particular, we believe we may not have produced this work without Saarinen and Beullens' comments on the `[pqc-forum]`.

attack was discovered soon after [38], which may push ALTEQ to update their parameters, which will be done regardless of the attack since their initial parameters were initially miscalculated: some potential updated parameters were presented in [14] during the NIST 5th Standardization Conference, although the end parameters are still a discussion topic. In particular, [14] mentions that the ALTEQ team is working on using d-tensors with $d > 3$ to reduce the signature size, which *may* lead to drastically smaller signature sizes given similar security parameters λ. At the moment the paper is being written, all of this happened in barely a year: everything here is *very* recent. Despite those attacks, the confidence in the ATFE problem remains as strong as ever: it is TI-complete (Tensor Isormorphism) and this class of problem has been deemed difficult by different cross-disciplinary research fields for several decades. Furthermore, *every* ATFE instance can be used in ALTEQ: thus, barring implementation technicalities (random number generation, sampling and so on) or protocol security (GMW-FS), the ATFE instances and the ALTEQ instances map one-to-one. This contrasts with cryptosystems where the theoretically hard problems do not match the cryptographic instances.

The work here is more practical. We do not revolutionize the world of theoretical cryptography, but we open the door to more advanced algebraic primitives to be studied and provide more practical adaptations of the base ALTEQ cryptosystem to reduce the gap between the theoretical cryptosystem and their hypothetical real-life deployments. We focus on ALTEQ simply because we are more familiar with it, but the core reasoning of our work can extend to LESS [6] and MEDS [17], or any group action-based cryptography. As "security proofs" tend to be outside our domain of expertise[4], we tend to rely on heuristic analysis instead and thus we name our work Dangerous Variations of ALTEQ (DVA)[5], which provide trade-offs in computational speed and data sizes, and could then be potentially better suited than ALTEQ for exploratory research whether on the protocol or on the mathematical primitive.

Our work has two objectives:

– To clearly encourage cryptanalysis, to be able to see a clearer picture in a somehow new area of studies: thus we take "risks", and titled the paper "DVA" and not "Secure Variation of ALTEQ". Many mathematicians from several fields outside of cryptography study the ATFE problem itself, and yet it still feels like a lot is left to discover [16], and ATFE applied to cryptography has *its own* specific problems. If no new cryptanalysis (sadly?) come forward, then maybe we just made ALTEQ much more competitive, which is somehow still a gain. The same can probably be said about the GMW-FS construction, although we are much less familiar with it. The paper is not exhaustive on *everything* that can be done as it focuses solely on presenting and justifying three types of variations (which may or may not mix), but aims to help

[4] But always willing to learn!
[5] And follow the trend in giving very distinct names to very similar cryptosystems.

kickstart as many as possible diverse research directions while keeping the contribution as simple as possible.

- To provide potential frameworks for ALTEQ to be sidegraded: ALTEQ deal with 3-dimensional tensors, but signature size can decrease drastically when considering d-dimensional tensors ($d > 3$): the matrix dimension n will lower to adapt to heuristic security analysis, and likely the size of the field q will lower as well while d has no impact on the signature size. This idea has already been mentioned in a recent poster [14] and within the ALTEQ team before the initial NIST submission, but practical deployment have been a concern and accurate security measure is complicated[6]. One *practical deployment* problem is that both computational complexity and data sizes to manage *outside the signature* (i.e, keys, internal computations) can be counted as polynomials of degree d: those increase drastically, even though the alternating structure slightly limits the explosion of the size. Some of the ideas we propose side-step the expensive costs or introduce structure to reduce the cost. In particular, the more qualities we can obtain in their practical deployment, the more attention we can attract to ALTEQ and similar schemes.

Contributions We propose essentially three research directions on ALTEQ, potentially interconnected, which for some already yields results, although we have to admit, of varying degrees of quality.

- DVA-GG (Generalized Graph) is more or less a fine-grained observation to group actions, where instead of considering one group action or the product, we observe the decomposition into smaller group actions just like the column decomposition in ALTEQ and construct less trivial uses to consecutive group actions. In particular, we show that we can leverage an extra group action in ALTEQ to protect against the seed collision attack of [9], discarding the need to use slower expanders to manage larger seeds in most parameters: as a direct consequence, this may allow ALTEQ to maintain the use of `AES-NI` for expansions and maybe gain efficiency by doing so. We can also leverage the structure of the group actions to construct fancy scenarios and exhibit some examples where the security of those constructions reduce to the ATFE problem or ALTEQ. Those constructions however, mostly serve as inspiration for exploratory research.
- DVA-PC (Precomputation) is a work on modifying the GMW-FS protocol used in ALTEQ, by using simple transformations to improve the practical application of the protocol. In particular, we heavily reduce the intermediate memory necessary to use the scheme: we could save more than 70 times the memory in the lowest parameters, or more than 400 times in the largest parameter. This also enables interesting trade-offs when cutting the scheme into two: an offline phase and an online phase. In particular, we have several choices of trade-offs combinations possibles. For example, the simplest option moves the d-tensor computation to the offline phase and leaves an online

[6] No official records, but a better analysis than “it is harder” may be required.

phase with a complexity *completely independent* of d and r, leading to an online complexity of K matrices multiplications/inversions excluding hashes and expansions. One extra option compress the offline storage to 3λ-bits of data per signature. Another option yields potentially large offline data per signature, but provide one of the fastest online signing speed in the history of post-quantum cryptography *in theory*. In any case, we also give the option to *completely remove from* ALTEQ the chance of having a failure due to column decompositions in the online phase, and also provide an option to partly solve the signing efficiency problem of [43]. Every option of DVA-PC maintain an equivalent online security to ALTEQ under the random oracle model, but we stress that there are security risks we are currently unable to properly quantify.
- DVA-DM (Dangerous Matrices) take a look at the matrices used through the whole ALTEQ scheme. ALTEQ uses the generic ATFE problem in their construction, as aside from a few technicalities (0 in the diagonal position of a column decomposition), *every batch of* ATFE *instances is a valid* ALTEQ *instance*. We take a step back from such a conservative approach and explore special structures, inspired by the historical development of lattice-based cryptography culminating to the standardized candidates [22,18] and beyond [19][7] and attempt to push some research on ALTEQ towards a similar direction. In particular, we provide several cryptanalytic targets, which were constructed as performance improving attempts ranging from an implementation view to more abstract ones on the arithmetic and algebraic operations. The goal is to encourage creativity and to help understand the different angle of attacks that could arise, as we believe an overly conservative approach could be good for standardization but not necessarily in setting incremental research targets.

While this paper focuses on ALTEQ for convenience as we are more familiar with it[8], most of the reasoning in this paper might not be primitive specific: it is possible that this work extend to further schemes.

Organization of the paper In section 2, we describe the ATFE problem and its variants, especially the ones that are relevant to us, briefly reintroduce the ALTEQ cryptosystem and the associated structures, which is in short the ATFE primitive used for a GMW-FS construction. In section 3, we present the point-of-view into multiple group actions that was somehow used in a “semi-hidden” way in ALTEQ, and introduce some potential constructs born from this point-of-view. In particular we explain a potential alternative countermeasure to [9]'s attack. In section 4, we first present our suggestion on reducing the intermediate memory of ALTEQ during certain steps of the sign/verify phase. Furthermore, we explain some direct applications, in particular the offline/online separation with precomputations, leading to the variants we mentioned earlier. In section

[7] [20] is an interesting outlier, which got rid of the structure.
[8] Yes, we definitely should study more schemes.

5, we present the different rationale that led us to the suggestions of our cryptanalytic targets, and how those, if secure, could improve ALTEQ's efficiency. In section 6, we finally list open questions that arise from this paper and encourage future research on ALTEQ and beyond.

Note on the paper history Initially a justified reject from PQC2024, and currently modified as an attempt to answer the reviewers' concerns as much as we could. We were advised to submit this work as a pre-print, hence here it is. Another reason to submit a pre-print rather than waiting until some hypothetical publication, is to contribute to research discussions *before* the round 2 selection process which could happen anytime now. In particular, some content here may help or inspire other schemes and not just ALTEQ.

2 Background

Here we give some reminders and notations about the essential information about ALTEQ that is necessary to understand this paper. Referring to [13] is recommended, but not necessary.

2.1 Basic notations

Mathematical notations

1. $q, n \in \mathbb{N}^*$: field orders and dimensions respectively. q is a prime power.
2. $\mathbb{F}_q$: the finite field of order q.
3. $\mathbb{F}_q^n$: vector space of $n \times 1$ vectors over $\mathbb{F}_q$.
4. $\mathrm{GL}(n, q)$: group of invertibles $n \times n$ matrices over $\mathbb{F}_q$.
5. For $u \in \mathbb{F}_q^n$ and $A \in \mathrm{M}(n, q)$, u^{t} and A^{t} denote their transposes.
6. For $a, b \in \mathbb{N}^*$, $\binom{a}{b}$ is the binomial coefficient.
7. For $a < b \in \mathbb{N}$, $[\![a, b]\!] = [a, b] \cap \mathbb{Z}$
8. For $m \in \mathbb{N}^*$, let $[m] = [\![1, m]\!]$.
9. Given S finite, $a \in_R S$ means a is a uniformly random sample from S.

Cryptographic scheme parameters ALTEQ system parameters are fixed in the following order:

1. λ the desired security level.
2. (n, q) the algebraic structure parameters, n, q as defined above.
3. (r, K, C) the GMW-FS framework parameters.

Gross notation abuse To save space and to reduce repetitive clutter we denote

- any collection $\{X_i\}_{i \in S}$ by $\{X_i\}_S$, it should be clear that i serves as an index of items X and is an element of S.
- $\mathsf{App}(\mathsf{S}, s)$ appends an element s to the end of an *ordered* list S.
- for S an *ordered* set of cardinal n, $||\mathsf{S}$ represents a concatenation with all its elements i.e $||s_1||...||s_n$. This is mostly used in hash function parameters where ordering is important, as a permutation change the hash output.

2.2 Trilinear forms and a natural group action on them

A trilinear form (TF) on $\mathbb{F}_q^n$ is a map $\phi : \mathbb{F}_q^n \times \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{F}_q$ that is $\mathbb{F}_q$-linear in each argument. It is *alternating* (ATF) if and only if
$\forall u, v \in \mathbb{F}_q^n$, $\phi(u, u, v) = \phi(u, v, u) = \phi(v, u, u) = 0$.
$\mathrm{ATF}(n, q)$ denotes the linear space of all ATFs on $\mathbb{F}_q^n$.

$\mathrm{GL}(n, q)$ naturally acts on $\mathrm{ATF}(n, q)$: for all $u, v, w \in \mathbb{F}_q^n$, $A \in \mathrm{GL}(n, q)$,
A sends ϕ to $\phi \circ A$, defined as $(\phi \circ A)(u, v, w) := \phi(A^t(u), A^t(v), A^t(w))$
This action defines an equivalence relation $\cong$ on $\mathrm{ATF}(n, q)$, namely
$\phi \cong \psi$ means $\exists A \in \mathrm{GL}(n, q)$ s.t $\phi = \psi \circ A$.

This is the main operation in ALTEQ, and the ATFE problems are built on this. Many variations of the ATFE problems have been defined [45], in particular the search version and the identification version:

– Search version: given $\phi \cong \psi$, find $A \in \mathrm{GL}(n, q)$ s.t $\phi = \psi \circ A$
– Identification version: given $\phi, \psi \in \mathrm{ATF}(n, q)$, is $\phi \cong \psi$ true?

For a less abstract representation, machine-wise we represent an ATF ϕ as

$$(c_{i,j,k} : 1 \leq i < j < k \leq n), c_{i,j,k} \in \mathbb{F}_q,$$

which requires $\binom{n}{3}$ entries (a *compressed* form), unlike generic TF that requires n^3 entries (a *decompressed* form).[9]

Higher dimensional tensors The principle generalize beyond dimension 3: in particular, the problem can be defined over d-dimensional objects, where the number of field elements of an alternating d-linear form (ALF_d or just ALF) is then $\binom{n}{d}$. The problem is in fact believed to be *heuristically* harder for fixed (n, q). We will continue to denote ALF_3 as ATF and ALF_d or ALF otherwise whenever d is considered to be possibly higher than 3. For the rest of this paper we will note ALF_3 for the ATFs used in ALTEQ, and ALF_d for $d > 3$.

2.3 Invertible column matrices

One of the key part of the current ALTEQ instantiation is the extensive usage of column matrices instead of generic matrices.

[9] A TF form is easier to use than an ATF for calculations due to array indexing: ALTEQ describes in their submission the detailed process, but here we assume the de/compression is done *within* functions and simply use a high level description.

Definition 1. *A matrix $C_i \in \mathbb{F}_q^{n\times n}$ is a column matrix, if it can only differ from the identity matrix by* ***at most one column*** *of index i i.e*

$$C_i = \begin{pmatrix} 1 \dots 0 & c_1 & 0 \dots 0 \\ \vdots \ \ddots \ \vdots & \vdots & \vdots \quad\ \vdots \\ 0 \dots 1 & c_{i-1} & 0 \dots 0 \\ 0 \dots 0 & c_i & 0 \dots 0 \\ 0 \dots 0 & c_{i+1} & 1 \dots 0 \\ \vdots \quad\ \vdots & \vdots & \vdots \ \ddots \ \vdots \\ 0 \dots 0 & c_n & 0 \dots 1 \end{pmatrix}$$

Obviously, the identity matrix is also column matrix for any index i, and for any column matrix C_i we have $c_i \neq 0 \Leftrightarrow C_i \in \mathrm{GL}(n,q)$.

Corollary 1. *$\forall A \in \mathrm{GL}(n,q)$, $\exists P \in S_n$ a permutation matrix such that AP is a product $\prod_{i=1}^{n} C_i$.*

In particular, if P is enforced to be the identity matrix, the decomposition $A = \prod_{i=1}^{n} C_i$ might not always exist, but if it does then it is unique. Its probability of existence is the same as having a LU decomposition with no permutation, thus for fields that are not too small the probability is overwhelmingly large.

In the above case, we denote A^{col} the set $\{C_i\}_{[\![1,n]\!]}$ s.t $A = \prod_{i=1}^{n} C_i$. Clearly, A^{col} takes as much memory as A (n^2 field elements) since we can discard the trivial columns. By abuse of notation, we then represent A^{col} by a single matrix, where its i-th column is in fact the i-th column of C_i (its only non-trivial column). We denote such a column A_i^{col}, i.e $A = \prod_{i=1}^{n} A_i^{col}$.

2.4 ALF$_3$s and group actions in algorithms

Here, we list the subroutines used to describe ALTEQ below. Note that for the understanding of this paper, it is not necessary to know *exactly* how the operations are done in practice, i.e how to compute $\phi \circ A$ given (A, ϕ). Everything can be summarized as long series of modular multiply-and-add by elements of A with elements of ϕ. We just need to know here that their computational complexities are asymptotically easy, i.e polynomial time and memory, but keep in mind that the group action cost the most, followed by the column decomposition, then by the matrix multiplication. The expanders have a non-zero cost, but as hardware acceleration exists (such as AES-NI), we do not have a correct comparison with the rest of the scheme (usually negligible except SHAKE).

Algebraic operations. Let S be a collection of indexes. Then we denote

– $\{\phi_i \circ A_i\}_S \leftarrow \mathsf{ActATF}(\{\phi_i, A_i^{col}\}_S)$
– $\{\phi_i \circ A_i^{-1}\}_S \leftarrow \mathsf{InvAct}(\{\phi_i, A_i^{col}\}_S)$
– $\{A_i^{col}\}_S \leftarrow \mathsf{ColDec}(\{A_i\}_S)$
  If A^{col} does not exist for a given A, a flag is raised to indicate failure.
– $\{C_i\}_S \leftarrow \mathsf{ColMul}(\{A_i^{col}, B_i^{col}\}_S)$
  Note that given A, B in column form, this return $C = A \times B$ and not C^{col}.

Randomness generation. Hashing is done using the Keccak (SHA-3) family of functions. Expanders are either based on AES or SHAKE. All functions use some bit-string as an input. Names are self-explicit:

- H is a hash function that takes an input of arbitrary length and output a binary string from $\{0,1\}^{2\lambda}$
- expCha is used for generating "unbalanced" challenges. (r, K, C) being fixed, it will output r indexes $\{b_i\}_{[r]} \in [\![0, C]\!]^r$ such that *exactly* $r - K$ indexes have $b_i = 0$ (and thus exactly K indexes have $b_i \in [C]$)[10].
- expATF outputs a random $\phi \in \mathrm{ALF}_3(n, q)$.
- expCols outputs a column decomposition A^{col} of some random $A \in \mathrm{GL}(n, q)$, such that A admits such a decomposition. This is done by directly sampling random columns and rejecting 0 values in diagonal positions.
- expSeeds outputs some specified number of seeds of any requested size,

2.5 The ALTEQ cryptosystem

We briefly present the generic pseudocodes of ALTEQ's setup, signature and verification processes in figure 1. Their description here is slightly modified compared to the original document [13], to make it simpler to understand and lighter to write without affecting the essential features of the scheme.

Difficulty of attacking the GMW-FS construction Aside from the ATFE primitive, line 9 of Vf is clearly what makes forging a signature hard, as it makes some form of "self-feeding" loop: Cha gives the positions of the ALF_3s, but Cha is given by the hash of the ALF_3s at said positions. A solution to the ATFE challenge must fit into the right position within the hashing parameters, and modifying the order changes the positions requested. Thus every position must feed into a specific order, and it does seem like the only way to forge a signature is to be able to solve the ATFE problem instances, assuming the encasing hash is secure. This naturally leads to force the number of combinations $\binom{r}{K}C^K$ to be above 2^λ to guarantee a λ-bit security.

The ATFE structure in ALTEQ The public key is in fact several ALF_3s where every each of them is within the same orbit: you can navigate from one ALF_3 to the other by using a group action by an invertible (column) matrix. Most of the attacks on ALTEQ would then focus on finding such a matrix. The signature provides matrices $M_{C\leftarrow B}$ s.t $\phi_C = \phi_B \circ M_{C\leftarrow B}$, knowing that $M_{C\leftarrow A}$ was first computed to generate the challenge. $M_{C\leftarrow A}$ is *never* given in the signature: *doing so automatically breaks the system.* As explained by Qiao [40], given two forms $\phi_A \cong \phi_B$, ALTEQ's security assumption relies on the difficulty of finding $M_{A\leftarrow B} \in \mathrm{GL}(n, q)$ s.t $\phi_A = \phi_B \circ M_{A\leftarrow B}$. Saarinen's attack used some $M_{A\leftarrow B} \notin \mathrm{GL}(n, q)$ (the zero matrix). This was patched using (half of) line 10 and

[10] In the original ALTEQ specification [13] (and in the code), the role of index C and 0 are swapped. We just inverted in this paper for conveniency.

ColDec[11] where $M_{A \leftarrow B} \notin \mathrm{GL}(n,q)$ are discarded by counting zeroes in $\{D_k^{col}\}$.

```
KGen
1:  $\mathsf{sk} \leftarrow_R \{0,1\}^\lambda$
2:  $\{\delta_i\}_{[\![0,C]\!]} \leftarrow \mathsf{expSeeds}(\mathsf{sk}, C+1)$
3:  $\phi_0 \leftarrow \mathsf{expATF}(\delta_0)$
4:  $\{\Delta_i^{col}\}_{[C]} \leftarrow \mathsf{expCols}(\{\delta_i\}_{[C]})$
5:  $\{\phi_i\}_{[C]} \leftarrow \mathsf{InvAct}(\{\phi_0, \Delta_i^{col}\}_{[C]})$
6:  $\mathsf{pk} \leftarrow (\{\phi_i\}_{[C]}, \delta_0)$
7:  return $(\mathsf{pk}, \mathsf{sk})$
```

```
Vf(pk, M, Sig)
1:  $\phi_0 \leftarrow \mathsf{expATF}(\delta_0),\ a \leftarrow 0,\ b \leftarrow 0$
2:  $\{c_i\}_{[r]} \leftarrow \mathsf{expCha}(\mathsf{Ch})$
3:  for $i \in [r]$ do
4:      if $c_i = 0$ then $a \leftarrow a+1$
5:          $D_i'^{col} \leftarrow \mathsf{expCols}(\mathsf{s}_a || \mathtt{salt} || i)$
6:      else $b \leftarrow b+1$
7:          $D_i'^{col} \leftarrow D_b^{col}$
8:  $\{\psi_i'\}_{[r]} \leftarrow \mathsf{ActATF}(\{\phi_{c_i}, D_i'^{col}\}_{[r]})$
9:  $\mathsf{Ch}' \leftarrow \mathsf{H}(\mathsf{H}(M) || \{\psi_i'\}_{[r]})$
10: $\mathsf{F} \leftarrow (\mathsf{Ch} \neq \mathsf{Ch}')$ or (bad $\{D_i^{col}\}$)
11: if F then return $No$
12: return $Yes$
```

```
Sign(sk, M)
1:  $\{\delta_i\}_{[\![0,C]\!]} \leftarrow \mathsf{expSeeds}(\mathsf{sk}, C+1)$
2:  $\phi_0 \leftarrow \mathsf{expATF}(\delta_0)$
3:  $\beta \leftarrow_R \{0,1\}^\lambda,\ \mathtt{salt} \leftarrow_R \{0,1\}^{2\lambda}$
4:  $\{\mathsf{s}_i\}_{[r]} \leftarrow \mathsf{expSeeds}(\beta, r)$
5:  $\{B_i^{col}\}_{[r]} \leftarrow \mathsf{expCols}(\{\mathsf{s}_i || \mathtt{salt} || i\}_{[r]})$
6:  $\{\psi_i\}_{[r]} \leftarrow \mathsf{ActATF}(\{\phi_0, B_i^{col}\}_{[r]})$
7:  $\mathsf{Ch} \leftarrow \mathsf{H}(\mathsf{H}(M) || \{\psi_i\}_{[r]})$
8:  $\{c_i\}_{[r]} \leftarrow \mathsf{expCha}(\mathsf{Ch})$
9:  $\mathsf{S} \leftarrow \{\},\ \mathsf{I}_\Delta \leftarrow \{\}$
10: for $i \in [r]$ do
11:     if $c_i = 0$ then $\mathsf{App}(\mathsf{S}, \mathsf{s}_i)$
12:     else $\mathsf{App}(\mathsf{I}_\Delta, i)$
13: $\{\Delta_{c_i}^{col}\}_{\mathsf{I}_\Delta} \leftarrow \mathsf{expCols}(\{\delta_{c_i}\}_{\mathsf{I}_\Delta})$
14: $\{D_i\}_{[K]} \leftarrow \mathsf{ColMul}(\{\Delta_{c_i}^{col}, B_i^{col}\}_{\mathsf{I}_\Delta})$
15: $\{D_i^{col}\}_{[K]} \leftarrow \mathsf{ColDec}(\{D_i\}_{[K]})$
16: if ColDec failed then go to line 3
17: $\mathsf{Sig} = (\mathsf{Ch}, \mathtt{salt}, \mathsf{S}, \{D_i^{col}\}_{[K]})$
18: return Sig
```

Fig. 1: The ALTEQ cryptosystem

Currently, there are two worries about ALTEQ's instantiations: that ϕ is *probabilistically* a weak key [10], or that matrices M are easy to find *regardless of the key*. The current parameters are set according to [10], and the latest poster parameters [14] takes into account [38]'s results. While those parameters might change depending on new attacks, this will not affect the work proposed here, but mostly the scale of its impact. In fact, if ALTEQ ever had to increase the

[11] [40]'s attack was published when ALTEQ's team did not publish ColDec yet. Without ColDec, [40] could have been devastating for performance reasons.

size of its parameters, then some of our proposed side grades are likely to be *even more* efficient.

Performance bottlenecks From our experimentations, it seems like the performance bottlenecks in ALTEQ comes from four points:

1. Computing group actions, from ActATF and InvAct.
2. Computing Ch from H, since the entry $\mathsf{H}(M)||\psi_1||\dots||\psi_r$ is very large.
3. Computing matrix products, from ColMul.
4. Computing column decompositions, from ColDec. Worse than ColMul, mostly due to modular inversion (currently use an adaptation of [44]).

We believe that tackling those 4 points while ensuring minimal loss of security would be a priority among those aiming to improve ALTEQ's performance. In this paper, we attempt to reduce or circumvent the cost of those operations.

Current data sizes of ALTEQ Following the description, we have

$$\begin{aligned}\mathsf{pk}_{\text{size}} &: C \cdot \binom{n}{3} \cdot \lceil \log_2(q) \rceil + \lambda \\ \mathsf{sk}_{\text{size}} &: \lambda \\ \mathsf{Sig}_{\text{size}} &: (r - K + 2) \cdot \lambda + K \cdot n^2 \cdot \lceil \log_2(q) \rceil + 2\lambda\end{aligned}$$

Note that the signature size has increased by 2λ to introduce a salt to patch [9].

3 DVA-GG: General Graph navigation, algebraic variants and patches to collision attacks

3.1 Rationale

For a visual representation of the navigation done in equivalence classes, we tend to jump directly from one point to the other without looking at the details. In fact, ALTEQ's use of the column decomposition divides the group action by an invertible matrix into a group action by n smaller invertible column matrices, which results in a faster overall group action operation, as visually shown in figure 2. While the decomposition (as implemented currently, i.e without permutations) does not always exist and the computation of the decomposition also takes non-negligible time, ALTEQ showed that in most cases we can just generate the decomposition from scratch, and enforce the associated composed matrix to be invertible.

We discuss in this section how this approach can lead to further research directions, sometimes beyond the simple computational acceleration. The variations we propose here, using this view, will be grouped as DVA-GG (Graph Generalization). Since a single of ALTEQ's group action computations is in reality n smaller group actions corresponding to a column decomposition, the idea here is to explore what would happen if we add even more parts, or observe more creative ways on how to use those parts: some parts could be public, some parts could involve third parties considerations. etc... For example, we can

$$\begin{tikzcd}
 & \tau_1 \arrow[r, "\tau_1 \circ C_2"'] & \tau_2 \arrow[r, dotted] & \tau_{n-2} \arrow[r, "\tau_{n-2} \circ C_{n-1}"'] & \tau_{n-1} \arrow[dr, "\tau_{n-1} \circ C_n"] & \\
\phi \arrow[ur, "\phi \circ C_1"] \arrow[rrrrr, "\phi \circ M"] & & & & & \psi
\end{tikzcd}$$

Fig. 2: ALTEQ's group actions using $M = \prod C_i$: the above "route" is faster.

– Attempt to craft an ALF_3-based counter to Beullens' recent seed collision attack [9], which we present as DVA-OTP.
– We can also consider a multi-user setting and leverage the algebraic properties of the primitives rather than a purely generic construction, which we present later as DVA-GM.

3.2 DVA-GG to DVA-OTP (One Transfert Point): Patching Beullens' attack with smaller seeds

Beullens' seed collision attack The modification we propose here from DVA-GG, that we define as DVA-OTP, is an algebraic patch to Beullens' seed collision attack that provide an alternative to expanding larger seeds. We first re-explain the seed collision attack of Beullens [9] to give an illustration of our technique. Beullens' attack work with the following steps:

1. Wait for a valid signatory to sign X signatures. Each signature uses r seeds, i.e r group actions (the round parameter) of the form $\phi_0 \circ B$ where B is a matrix expanded from a seed of λ-bits.
2. The attacker computes random group actions from chosen seeds and finds a collision between his computed group actions and the ones in the signatures with an effort of $2^\lambda/(r \times X)$ group actions.

Note that, unlike what is written in [9], we believe that not *all* collisions themselves are a problem. In particular, the r group actions contain $r - K$ group actions computed from ϕ_0 by the verifier itself and did not cause any problem before. The danger lies in colliding with the seeds used by the K chosen challenges, as revealing the seed, hence the matrix, reveals the secret matrices. A maybe tighter version of Beullens' attack works in the following way:

1. Wait for a valid signatory to sign X signatures. Each signature uses K matrices D_k for $k \in [K]$, i.e K group actions (the round parameter) of the form $\phi_0 \circ B = \phi_i \circ (D_k = \Delta_i B) = \psi_i$ where B is a matrix expanded from a seed of λ-bits for some $i \in [C]$.
2. The attacker computes random group actions $\phi_0 \circ B$ from chosen seeds and finds a collision with any $\phi_i \circ D_k$ with an effort of $2^\lambda/(K \times X)$ group actions as $D_k B^{-1} = \Delta_i$ reveals the key (or at least part of it).
3. Note that $X \times (r - K)$ group actions are computed from ϕ_0 as part of the signature/verification process, thus it is possible that the signatory helps to break his own key. This "internal collision" as mentioned by the MEDS/LESS teams can also be considered part of the attack.

This slight observation (r to K, internal collision) barely affects the efficiency of the attack (maybe by less than a single bit), so it is likely that for simplicity this detail was known but omitted[12].

The main takeaway is that the seed collision attack of Beullens is first and foremost an ALF_3 collision attack by colliding $\phi_0 \circ B$ and a ψ_i from a signature. The current patch used by ALTEQ, MEDS and LESS is the following: instead of expanding the matrix B by a seed s, the matrix B is expanded by a larger seed

$$s_{new} = (s_{old}|\texttt{salt}|r_{id})$$

where s_{old} is the original λ-bits seed, `salt` is a bit string given in the signature, and r_{id} corresponds to the position in the r group actions in the signature. Assuming that each signature gives a different salt (so accumulating signatures does not work), then the effort of an attacker is pushed back to 2^{λ} as r_{id} *theoretically* forces each pool of matrices per position to be unique assuming $\mathrm{GL}(n, q)$ is large enough.

Another way to patch this attack is to increase the seed size: intuitively, we believe that for a seed size l, $2^l/(K \times X) > 2^{\lambda}$ should be enough to ward off the attacks. If we follow the NIST recommendations and just admit a limit of $X = 2^{64}$ of signatures, then $l > (\lambda + 64 + \log_2(K))$ should be enough to ward-off the attack if we ignore the internal collision probability. This solution however increases the signature size more than a salt, namely $(r - K)l$-extra bits compared to a salt of 2λ or a flat number[13]. l can also be reduced with r_{id}, which does not increase signature size: it basically eliminates $\log_2(K)$ from consideration. Matrices usually take most of the signature size so either solution seem to be roughly equivalent. However, there are implications in appending a salt (conveniently noted `salt`) and a position tag r_{id}: permuting samples. Following the same logic, we could use `salt` but remove r_{id}: either the seeds and/or `salt` would have to be increased in size to compensate the loss of r_{id}. This has more implications later in the paper, namely in section 4 concerning data management, but we skip this detail for now.

Expanding from $\phi_{1,\dots,C}$ when $\mathrm{GL}(n, q)$ or the orbit is small enough Beullens' attack can slightly change in behaviour if we expand directly from $\phi_{1,\dots,C}$ instead of ϕ_0. It is very unlikely that this will lead to an ALF_3 collision, but the advantage of that approach is that colliding with any of the $r - K$ parts of the signature could reveal a secret, and the K parts would still be relevant: in particular, learning the link $\phi_i = \phi_j \circ M$ where $i, j > 0$ strongly simplifies forgery attacks

[12] Thus seeds can be re-used as long as they are never linked with any of the K matrices given per signature. However, if we go towards a sample-and-retry approach to avoid collision with the K positions, we might as well use the sample-and-retry approach of [43] to reduce both signature size and group action costs.

[13] While we are not very familiar with MEDS and LESS, both other submissions seemed to have opted for different salt sizes.

by effectively reducing the security parameter K to $K-1$, even without the secret key. So how likely is an ALF$_3$ collision? Well this can occur when $\mathrm{GL}(n,q)$ is small enough such that seed can expand into M_1 which would equal a product of $\Delta_k M_2$, or more interestingly, if there exists $M_1 \neq M_2$ such that $\phi_i \circ M_1 = \phi_i \circ M_2$. In the latter case even when $i = 0$, we found a part of the automorphism group which may lead to very interesting attacks on ALTEQ and is clearly outside this paper's scope. Thus, any work that studies the automorphism group of the ATFE problem with $\mathrm{GL}(n,q)$ could be vital for the security of ALTEQ (and important for group action cryptography): in practice, it is not $\mathrm{GL}(n,q)$ that needs to be large, but the orbit. We briefly mention the above to encourage the academic community to dig into it, but we do not do it here.

Patching the attack using DVA-GG In ALTEQ, patching the attack with meant they had to give up on the `AES-NI` hardware instructions set and rely on SHAKE for expansion: this decreased their verification speed by 5% to 20% depending on the parameter set as expansion can be costly. Thus we use DVA-GG to present an alternative to the larger seed/single expansion approach by modelizing the group actions, or point-to-point translations in the graph, with more intermediate steps. The salt being unique per signature means that the set of ALF$_3$s to collide with is unique per signature, and the approach presented previously was made by changing B and not ϕ_0 in the ALF$_3$s of the signature. We propose to instead change the starting point ϕ_0 per signature, using a second translation to ϕ'_0, a *a transfer point*, i.e

- For each signature, we take a matrix A and generate $\psi_0 = \phi_0 \circ A$.
- Then every round has $\psi_i = \psi_0 \circ B = \phi_0 \circ AB$ instead of $\psi_i = \phi_0 \circ B$.
- The amount of possible ψ_i to collide with is thus extended by the possible A, just like the possible `salt` used by an extended seed s_{new}.

Note that, just like the previous approach, if n, q are too low due to adapting to ALF$_d$ for a given security λ, there might not be enough distinct matrices to either expand or multiply to (i.e $\mathrm{GL}(n,q)$ is too small) but this is a consideration that does not apply currently for most parameters for 3-tensors (i.e ALTEQ). In a way we generalize the set of challenges within the equivalence class by offering an additional translation and give a representation in figure 3 for the transfer of one single point for all ALF$_3$s.

In this example of DVA-OTP, we consider to only replace the salt to give the following extended seed

$$s_{\mathsf{DVA\text{-}GG}} = (s_{old} | r_{id})$$

and as long as $s_{\mathsf{DVA}-GG}$ is less than 256-bits then we can use `AES-NI` instructions to expand the seeds. Note that this will not apply for security category 5 parameters when $\lambda = 256$ as long as r_{id} is appended, but we could argue that a smaller input size also makes SHAKE slightly more efficient. The trade-off is then the following:

- For the signatory, K more *special* matrix multiplications, one *special* group action, but r lighter and faster expansions.

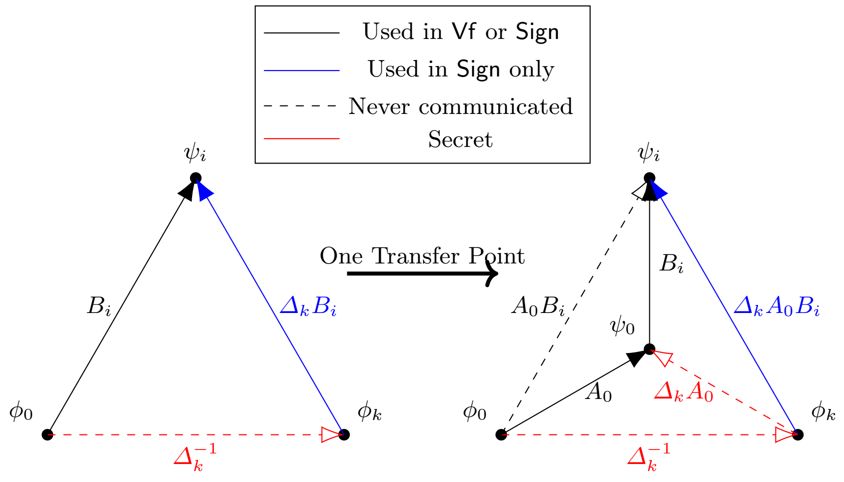

Fig. 3: ALTEQ to DVA-GG OTP

– For the verifier, one *special* group action, but r lighter and faster expansions.

We say *special*, as neither the complexity of the multiplication nor the complexity of the group action by A needs to be as hard as the random case: the sole purpose is to avoid collision. For this purpose only A does not need to be sampled uniformly in $\mathrm{GL}(n, q)$: if we denote $\mathcal{A}$ as the set containing the chosen matrices A, that we will call *the salt matrices*, we could choose $\mathcal{A}$ as monomial matrices (known as scaled permutation matrices).

Note that since A is publicly given by signature, it is enough that A is a diagonal matrix as long as the size of $\mathcal{A}$ is larger than the size of `salt` that we attempt to replace. In particular, for ALTEQ we observe that n and q are large enough such that distributing the `salt` bits as additive elements to the diagonal coefficients of the identity matrix is enough: we do not need to call any expander to sample from $\mathcal{A}$, and we can even limit the cost of arithmetic operations due to the low-size integers at particular positions. This gives raise in algorithm 4, that transforms a salt into a salt matrix. Note that algorithm 4 is not the only way to transform a salt into a salt matrix: we could also stack the first coefficients to have maximal size and leave the rest to ones. If the lack of mixing seems scary, we could always expand the coefficients with `AES256-CTR` and add permutations into the mix: typically any salt with less than 256-bits would enable a “fast” expansion through `AES256-CTR` whenever `AES-NI` is available.

While the group of diagonal matrices are likely an easy-to-break group [16], this is not a problem here: the main concern is whether this modification gives *less* security than adding a public salt to the seed expander, and when is this modification more efficient than adding a salt to the expanders.

First, whatever the algebraic structure given by the salt matrix, it is unlikely that this leads to an attack that is faster than the current procedure with

```
ConvertSalt(salt) (one basic method)
1 : $b \leftarrow \lfloor \log_2(q)/n \rfloor$
2 : if $b * n = \log_2(q)$ then $r \leftarrow 0$ else $r \leftarrow 1$
3 : $A \leftarrow Id_n, s \leftarrow$ salt
4 : for $i \in [1, n-r]$ do
5 :     $A[i,i] \leftarrow A[i,i] + (s \mod 2^b)$
6 :     $buf \leftarrow \lfloor s/2^b \rfloor$
7 : if $r = 1$ then $A[n,n] \leftarrow A[n,n] + s$
8 : return $A$
```

Fig. 4: Transforming salt to a salt matrix

the normal salts `salt`, which is: expand a seed with the known `salt` and test for group action collision. While there are computable algebraic relationships between ALF$_3$s that use the same seeds and the same positions for the $r - K$ seeds given publicly in the signature, it is not clear how this would enable a significantly faster attack concerning the K matrices that involves the secret key.

Second, the larger is the number of rounds, the more efficient this transformation is likely to be, especially when r is over the hundreds and one extra *special* group action should add less than 1% of computation time (at least in non-vectorized implementations) when experimentally having SHAKE instead of `AES256-CTR` using `AES-NI` lose us $> 5\%$ time. Note, that we can still expand this extra group action from $\mathrm{GL}(n, q)$, as long as the cost of the extra group action is lower than the performance loss due to expanding larger seeds, the strategy remain valid.

Relying on the non-commutativity for security Let us give a more algebraic point-of-view of DVA-OTP. The security here relies on the assumption that recognizing ALF$_3$s that used the *same seed* but *different salt* is hard, thanks to the group action computation being non-commutative. For A a chosen salt matrix, $B \in \mathrm{GL}(n, q)$ being secret and generated on a seed, The K matrices used for signing have form $(\Delta_i \times A \times B)$ and

- On top of attacking $(\Delta_i \times A \times B)$, $(\Delta_i \times A \times B)^{-1}$ is available.
- Similarly, $(\Delta_i \times A \times B)^t$ is also available for an attacker to play with.

Since the new public knowledge is A, the problem now is to go back to the original ALTEQ equation, or rather the initial ALF$_3$ that would allow the original attack to be restored, i.e to find $(\Delta_i \times B)$ or $\psi = \phi \circ (\Delta_i \times B)$. For randomly sampled $A \in \mathrm{GL}(n, q)$ this is probably as hard as ALTEQ itself, but we used diagonal matrices as our example, which can be commutative in certain cases (like whenever A or B is symmetric, which is unlikely for the current ALTEQ parameters). It is probably safer to use less trivial sparse matrices such as low-weight cyclic matrices, but to encourage cryptanalytic targets we keep the diagonal matrices for now. A graphical approach on breaking DVA-OTP, is to consider a special identification problem that we show in figure 5 where ψ_{B_1} are the ALF$_3$s computed by an attacker in the original attack.

$$\begin{tikzcd}
\phi_i \arrow[rr, "\Delta_i A B_2"] \arrow[dr, dotted, "\Delta_i A"] \arrow[dd, dotted, "\Delta_i"'] & & \psi_{AB_2} \\
& \psi_A \arrow[ur, dotted, "B_2"'] & \\
\phi_0 \arrow[ur, "A"] \arrow[rr, "B_1"] & & \psi_{B_1}
\end{tikzcd}$$

Fig. 5: DVA-OTP's “special" identification problem: $B_1 = B_2$?

If this “special” identification problem requires *less than a group action computation*, then the protection provided by this variation to Beullens' seed collision attack is weaker than expanding with a salt. If it requires at least *one* group action, then we can assume its security is equal to the original countermeasure of expanding larger seeds containing salt and positions, since testing equality of a random group action operation can be done after fixing a salt value just like fixing a matrix A.

***TODO**: implementation test* In theory, for r large enough we should be more efficient as long as `AES-NI` is available, but we have not yet completed an implementation to test our theory. The problem is r being large usually means we target a larger security level, where the seeds are large, and hence for seeds larger than 256-bits our proposal might not be relevant. We will attempt to publicly release a test implementation when we can.

3.3 DVA-GG and splitting seeds into distinct algebraic components

In the case that the basic DVA-OTP is insecure no matter what matrix set $\mathcal{A}$ we use, we still two options to use to keep using faster expanders that requires lower value seeds.

- Option 1, cut the expansion into multiple calls of the faster expander: for example expand s_1 for the lower bits and s_2 for the higher bits of the matrix coefficients, such that

$$s_1 = (s_{old}|\texttt{salt}_1|r_{id}),\ s_2 = (s_{old}|\texttt{salt}_2|r_{id}),\ \texttt{salt} = (\texttt{salt}_1|\texttt{salt}_2)$$

  We believe this should be safe algebraically, as the opposite could imply we can separate and simplify the ATFE problems into lower and higher bits

parts. Note, this above example makes sense if and only if s_1 and s_2 are less than 256-bits and calling `AES-NI` intrinsics twice for `AES256-CTR` is faster than using SHAKE once. This process can be generalized into k splits for $(n^2/k)\log_2(q)$-bits per expander, as long as we remain more efficient than a single SHAKE call for $n^2\log_2(q)$-bits. The value `salt` does not have to be the only cut value either, as s_{old} can also be cut. This is useful for whenever s_{old} is also too large for the fast expanders (256-bits, for example).
- Option 2, deepen DVA-GG even further for algebraic variants of Option 1.

To exploit DVA-GG even further, we make note that DVA-OTP only offered one point for every ALF_3s. With split seeding, we can in fact offer multiple transfers *per* ALF_3. For example, suppose every position splits into 2 group actions, using the LU decomposition:

- s_1 expands into a lower triangular matrix L, and is the first group action.
- s_2 expands into an upper triangular matrix U, and is the second and last.

Since almost every matrix of $\mathrm{GL}(n,q)$ admits an LU decomposition, this method allows a safe double expansion with close to the same arithmetic cost as a full matrix. The L, U matrices can also be decomposed/generated in column form, so theoretically there is no loss in efficiency either.

As shown earlier in figure 2, ALTEQ in fact already performs a splitting into n transition points by using the column decomposition of their invertible matrices: each column is considered a group action, and their code reflects that as their group action function is called n times per matrix, with a different index parameter and a different column. Thus we could in fact split the seeds to expand into several columns, by option 1 for all columns, or by choosing different split seeds per column. Security-wise, we do not see any particular issue: the $r-K$ seed expansions in the signature are public anyway as used by the verifier, and the K matrices are not just the result of an expansion but are the result of a matrix product thus the multiplication by the secret key (hopefully generated from a more secure method) should hide any poorly structured randomness from a smaller seed that an attacker wish to exploit.

3.4 DVA-GG potentially opening split structures

Beyond attempting different counters to Beullens' seed collision attack, DVA-GG warms-up the exploration of diverse matrix decompositions. Recall that ALTEQ already uses a matrix decomposition to improve efficiency, and DVA-OTP uses a public/private split of a group action to patch seed collision vulnerabilities through an algebraic decomposition. This approach can be slightly altered to accelerate signatures and verifications:

- Instead of using an easy to compute public part and a hard to compute secret part, we can instead use a *"not-so-easy"* to compute public part and

"less-hard" to compute private part. For a two-part DVA-OTP, imagine $(\varDelta_i \times A \times B)$ where the public part A could be set as $\mathrm{GL}(n, q)$ but B is a smaller, easier to compute set for group actions than $\mathrm{GL}(n, q)$.
- Since the computation of A is done only once, a sufficiently easier to compute set of matrices B could lead to a performance increase even in the presence of a low value r in the (r, K, C) parameters, i.e a low amount of rounds.
- While there is a distinct public and private part, the private expansion can re-use information of the public part and likewise, both salt matrices A and expanded matrices B can be split into multiple parts (such as n column matrices).

Note, that if we maintain the secret matrices $\varDelta_i$ as random elements of $\mathrm{GL}(n, q)$, then the "faster" matrices B are mostly useful for the verifier for $r - K$ group actions, thus for any choice of parameters (r, K, C) this thought experiment is only useful when K is much lower than r, which should usually be the case when aiming to minimize signature size anyway.

Whether the security approach of DVA-OTP remains valid, an "easier to compute" B leads to two new potential weaknesses we need to be wary of:

1. Leak attacks in the presence of many signatures. Since there is full knowledge of A in $(\varDelta_i \times A \times B)$, it is possible that information on $\varDelta_i$ is leaked through an accumulation of many signatures using the same $\varDelta_i$ if the set of matrices B is picked from introduce leakage.
2. Direct recovery of B through standard attacks: since $\psi_0 = \phi_0 \circ A$ is public, an attacker can just directly attack $\psi_i = \psi_0 \circ B_i$ where B_i is likely to enable faster heuristic attacks: known zero positions would simplify the algebraic systems to solve for example.

The first point might be a bit premature to consider, as NIST itself fixed a hard limit of 2^{64} signatures per key for their candidates *and* is also willing to consider systems with a lower limit which in our opinion enable the consideration of cryptosystems that leak but have limited leakage[14]. It is also better to research leak attacks that do not even exist for ALTEQ yet, *after* the second point has been considered. The second point is more interesting, as it encourages to craft efficient algebraic structures while considering different types of attacks. Some attacks are q insensitive and only depend on n, some attacks depends on both n and q, so it is possible that for some parameter sets there is an alternative to $\mathrm{GL}(n, q)$ that does change the *minimum* security level for *all* attacks. For example, suppose we measure the security by an attack α and an attack β, where the security for α is 128-bits and the security for β is 190-bits: we could introduce then an algebraic structure, for which the security by α is unchanged but the security by β lowers to 130-bits. The overall security level would be unchanged and remain at 128-bits. In particular, such scenarios could arise from changing

[14] "NIST asks for public feedback on a version of SPHINCS$^+$ with a lower number of maximum signatures" [35]. But we might be over-interpreting.

the ALF_3 of ALTEQ to ALF_d: some attacks may be disabled entirely, while others might see an exponential increase in complexity, or may stay unchanged such as brute-forcing the seeds. However, diverging from $\mathrm{GL}(n, q)$ only makes sense if there is some form of performance increase, mainly implementation-wise. A whole discussion on alternatives to ALTEQ's column decompositions of $\mathrm{GL}(n, q)$ is discussed in section 5.

3.5 DVA-GG to DVA-GM (Group Messages): Signing for a group

The following part is a usage of the DVA-GG point-of-view to inject certain fancy capabilities into the ALTEQ scheme which may or may not be plugged into more advanced cryptographic constructions. In DVA-OTP, the signature contained a salt `salt` extended into a "salt matrix" S, where the matrix created a transfer point ψ_0 that determines exactly the rest of all subsequent following computation results of $\psi_1, \psi_2, ...$ In this part, we exploit DVA-GG by inserting points just like DVA-OTP but to create different constructions, which could be arguably be done without the properties of the graph. The point here is to show how the algebraic structure *can* be leveraged instead of generic hash functions: there are probably better cryptographic constructions out there that serve the same purpose[15], but the aim is to *maybe* inspire new usages of the primitive to people more familiar with super fancy cryptographic constructions.

A hospital's waiting room Let us present a simple construction using the same trick as in DVA-OTP, i.e a salt matrix S generated involves s_{id} concatenated to the salt, or even with the message: the point is to force the salt matrix generation to be involved using the identification string s_{id}, but not *just* s_{id}. This generation has to be one-way, i.e *do not use the generation shown in algorithm 4*. Our hospital's waiting room proceeds as follows:

1. A patient arrives with his own identification string s_{id}, or is provided one by the hospital.
2. When the hospital wants to call the patient, they provide a signature where the salt matrix is
$$S \leftarrow \mathsf{expCols}(r||s_{id})$$
where r is chosen randomly (or not) by the hospital, and publicly provide a signature by omitting S, s_{id}, but provide r.
3. The patient can reveal S, every party can verify that the signature is correct with the salt matrix S, and s_{id} is not revealed. The same patient can be called several times, or never called at all: the hospital gets to decide.

This construction can also be achieved without an algebraic component by using a new random bit string as a salt instead of a salt matrix, but we aim to showcase that algebraic properties can be leveraged instead. No new calculation capacity

[15] We are not familiar with advanced cryptographic constructions, but we are welcoming any new knowledge.

is necessary: column expansion is already required to verify a signature, so if $r||s_{id}$ is lower than a seed we could re-use the original function. The same could be said for an expanding a new random bit string, but this depends on the functions used (SHA3, AES, etc...). Note that variations exist, which prompts caution:

- without r, s_{id} has a unique use.
- if s_{id} is chosen by the patient, r needs to be mandatory as there could be otherwise be a risk that a malicious patient attempt to recover the hospital's keys by choosing the intermediate ALF_3/salt matrix. Note that we do not need to limit ourselves to *one transfer point*, so we can combine with another layer of DVA-OTP to avoid this scenario, or just use extended expansion with salt/position.
- The hospital can also send a signature nobody can verify, or have a dishonest patient that in fact do not want to step forward.

This hospital's waiting room scenario illustrates some semi-algebraic variant of ATFE problem, represented by the shape of its "incomplete" signature in figure 6: given a correct signature for DVA-OTP but without the salt matrix S, can you guess S, i.e the transfer from ϕ_0 to ψ_0? In a sense, this is a "reverse" ALTEQ, where you have the signature but are lacking some public key information. This could be harder than ALTEQ, since the challenge generation is not algebraically linked to the ATFE problem and the ψ_i are not part of the signature. If the final ALF_3s $(\psi_1, \psi_2, \dots)$ are given, then the problem is purely algebraic and reduce to a pure ATFE problem between ϕ_0 and ψ_0. In particular, giving a large amount of signatures with the same ψ_0 might not help solve the problem: ψ_0 can always be guessed through inversion if the ALF_3s ψ_i are given.

$$\begin{tikzcd}
 & & \phi_0 \arrow[d, dotted, "S?"] & \phi_1 \arrow[dd, "M_{1,4}"'] & & \phi_2 \arrow[dd, "M_{2,6}"] \\
 & & \psi_0 \arrow[dll, "s_1"'] \arrow[dl, "s_2"] \arrow[d, "s_3"] \arrow[drr, "s_5"] & & & \\
\psi_1 & \psi_2 & \psi_3 & \psi_4 & \psi_5 & \psi_6
\end{tikzcd}$$

Fig. 6: DVA-GM, hospital's queue: what is S given $(s_1, s_2, s_3, s_5, M_{1,4}, M_{2,6})$?

Forcing a conversation order in a group chat, protect chat history Let us consider a group chat, where everybody signs their own messages, and the server just accepts and distributes the *correctly signed* messages to the recipients and keep very limited internal data. The setting is the following:

- Every user generates their own public and private key, for the group chat, and will use those to sign their self-generated messages. We assume the public keys have been shared.
- The server initially and publicly attributes a matrix $M_i \in \mathrm{GL}(n, q)$ for each user, and a matrix $G \in \mathrm{GL}(n, q)$ for the group chat: this information can be given to the public.

Every time a user i sends a message, the following process happens:

1. The message must be signed with the public key of the sender, with a "transfer point" ψ_0 specific to the sender just as in DVA-OTP, using $G \times M_i$ or just G such that $\psi_0 = \phi_0 \circ G$, but do not need to send it as part of the signature: the server and every other user can verify themselves that the message signed with the sender's public key is valid as G, M_i are public data (and thus can compute the correct intermediate ALF_3 from the sender's public key).
2. The server redistributes the signed message to the other users, but G is updated to $G \leftarrow G \times M_i$ and M_i is updated according to the content of the new message and its previous data. This could be a new expansion using the previous M_i, the message and the previous group data. We could also choose to not update M_i, but then G is independent of the message and signature contents.
3. Note that at all times beyond the setup, M_i, G needs not to be sent: since every user has their own local copy, they could just update G, M_i themselves.

The above process allows to force the integrity of the all messages in all devices: a somehow simple representation is given in figure 7. For the sake of simplicity, we also ignore all details pertaining to encryption of the messages, establishing trust between users, the server and the sharing of the public keys per user, etc. We could verify from scratch, that the whole conversation has been unaltered by a single party, as every user has the capability to recompute all the successive transfer points from the content of the messages, and the order in which they were sent. It is possible to delete previous messages if the status is stored, but then we can only check the integrity of the message history from before the deletion and after the deletion but not consecutively between the two parts.

Note that we were intentionally vague on how M_i is updated, as it depends on what properties we would like to enforce:

- M_i is never updated, only G: any user can freely modify their own message in their device. This will not show on other people's devices, and there might not be any proof of tampering. If the message disappears, other users should be able to guess through G that *something* must have been there.

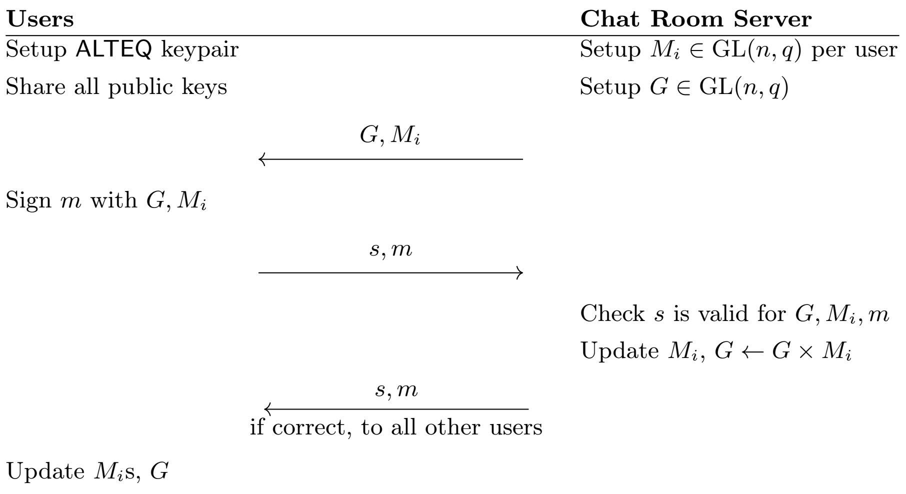
Users
Chat Room Server
Setup ALTEQ keypair
Setup $M_i \in \mathrm{GL}(n,q)$ per user
Share all public keys
Setup $G \in \mathrm{GL}(n,q)$
G, M_i
Sign m with G, M_i
s, m
Check s is valid for G, M_i, m
Update M_i, $G \leftarrow G \times M_i$
s, m
if correct, to all other users
Update M_is, G

Fig. 7: DVA-GM, a (heavily simplified) group chat

- M_i is updated with its own message content, but *after* the signing process: it becomes possible for other users to predict the transfer point of this user, if he is bound to be the next one to write a message and its message pattern is predictable.
- M_i is updated *before* the signing process with the message: unless the other users know exactly what the message is going to be, they should not be able to guess the transfer point.
- M_i is updated simultaneously and tied to the signature/message content: not only predicting the next step is hard, but modifying a message breaks the verification chain.

With this process, from the starting M_i and the starting G, any group user can check the integrity of his chat history by checking that all successive matrix multiplications lead to correct transfer points, correct signatures for each public key at every step of the conversation for all messages. Note, that this forces asynchronous messaging: if several users send a message to the server at the same time, only *one unique message* will be accepted to update G and M_is and the others discarded, and render invalid every other message sent simultaneously (before receiving the new message). This property also implies that a user cannot communicate with the other users if the status matrices are not up-to-date (and possibly checked to be correct according to all new messages), or at least cannot prove his messages are correct within the group at this point in time. In the example construction we just described, the server is quite passive. It could play a more active role, as forcing certain updates to M_i and G per message so that consecutive messages from the same user have to wait for server updates before being treated. All-in-all, the challenge is the following: given a recording of successive messages, can a malicious external party insert false messages undetected within the group chat, corrupting chat history?

4 DVA-PC: precomputations for more practical usages

4.1 Rationale

The GMW-FS construction used by ALTEQ, for which we do not go into details, was built on security proofs on top of another. There are however practical considerations that the theoretical constructions do not usually consider, and one of them is intermediate memory. One of the main considerations when setting parameters using the GMW-FS framework is to either minimize the size of the signature, largely dominated by the choice of K, or the public key, largely dominated by the choice of C. As a result, the round number r is often considered last, even though it is the main factor in computation time. In particular, the intermediate memory used by the challenge computation costs a lot of memory: we need to hash r ALF_3s, and for any parameter $r > C$ this can be problematic considering the already large size of each ALF_3. This will become worse if we move to ALF_d.

Thus, we retrace the computation steps that leads to the challenge and come up with a simple solution:

1. The seeds s_a are expanded into matrices $D_i'^{col}$.
2. The matrices $D_i'^{col}$ are used to compute ψ_i' from ϕ_0.
3. Those ψ_i' are used with the hashed message $\mathsf{H}(M)$, to compute Ch'. This is one of the speed and memory bottlenecks of ALTEQ.
4. Since ψ_i' are heavy, why not give $h_i = \mathsf{H}(\psi_i')$ instead?

This simple transformation should be as secure as ALTEQ under the "random oracle model"[16]. We call this second variation DVA-PC (or DVA-PC_0 as Precomputation 0), and present the only change in figure 8. The rest of the section is to show this extra, mostly inexpensive step, has uses.

In theory, this looks like an extra computation step, but in practice, we see at least two advantages that can be leveraged performance-wise:

- Each hash of an ALF_3 can be computed independently on a parallel manner. The final output is a hash of then much smaller data, which can directly translates into a speed improvement in both signing and verifying time.
- The intermediate memory necessary to compute the challenge can be essentially reduced to one ALF_3 and r hashes. Due to the large size of the ALF_3s, this is often less than two ALF_3s. For vectorized/parallel implementations, we can limit the number of ALF_3s to be pre-hashed simultaneously by a

[16] This was pointed by a reviewer, and we believe it is correct, but sadly we also believe this does not give any form of protection against the attack risks we later explain.

```
$\mathsf{Sign}_{\mathsf{DVA}}(\mathsf{sk}, M)$
⋮
7 :  $\mathsf{Ch} \leftarrow \mathsf{H}(\mathsf{H}(M)||\{\mathsf{H}(\psi_i)\}_{[r]})$
⋮
```

```
$\mathsf{Vf}_{\mathsf{DVA}}(\mathsf{pk}, M, \mathsf{Sig})$
⋮
9 :  $\mathsf{Ch}' \leftarrow \mathsf{H}(\mathsf{H}(M)||\{\mathsf{H}(\psi_i)\}_{[r]})$
⋮
```

Fig. 8: DVA-PC$_0$: Precomputations, version 0

number of ALF$_3$s smaller than the number of rounds r. In table 1 we compare the size of an ALF$_3$ according to the new parameters of [14], and the number of hashes of size 2λ-bits to reach an equivalent entry size: in particular the input size is 77 lower for the lowest security level, and the input size savings tend to increase with higher security levels or tensors degrees[17] making the option of replacing ALF$_3$s by hashes more attractive.

– This ease-up the communications if we wish to delegate the computations of the group actions to an external device: ALF$_3$ and their group action computations takes the most time and intermediate memory by far. It could then be interesting to delegate the computation of such items to external devices, through trust must be established first [24], and we do not consider this problematic for now: an extra computational device connected through USB or a local network could do the trick[18]. In such a case, the exchanges between the two parties are basically seeds (or matrices) and hashes, after a setup process to communicate the ALFs: since only the hash is necessary for the computation of the challenge, transmitting the ALF$_3$ or ALF$_d$ becomes unnecessary. In particular for security level 1 ($\lambda = 128$), a hash of an ALF$_3$ is 2λ-bits and thus at least 77 times lighter than the ALF$_3$ itself.

A note on memory costs Very recently, we were told to have a look at [28]. We are grateful to be pointed towards this presentation as we acquired new knowledge from the slides. Essentially, ALTEQ was not tested in [28]'s target platform because of its huge memory cost. While DVA-PC$_0$ can partly solve this problem, it might not be enough itself: the GMW-FS construction in its core seems particularly ill-suited for this as the FS step takes a lot of memory. Targeting the machine advertised by [28] was never a target of ALTEQ's conceptors, but it is doable, at least for security level 1 in our opinion: the first step to change would be the (r, K, C) parameters by reducing r and C but increasing K which would

[17] For 4-tensors, n, q would likely be lowered compared to their ALF$_3$ counterpart by matching the resulting increase in heuristic security. Thus, those values are clearly overestimated but serve as a reference.

[18] Our prior PQC2024 submission considered an online server users would rely on to, but if this online server is malicious this would cause havoc as pointed out by a reviewer. For now, we are limiting the scope of the third parties unless a provably trustable setup comes out.

NIST security level	1	3	5
Estimation of $n, \log_2(q)$	18, 24	27, 21	39, 24
Bitsize of one ATF (3-tensor)	19584	61425	219336
Equivalent number of 2λ-bits hashes	77	160	429
Bitsize of one 4-tensor	73440	368550	1974024
Equivalent number of 2λ-bits hashes	287	960	3856

Table 1: Size of entries of the challenge generation (excluding $\mathsf{H}(M)$)

lead to an increased signature size but decrease intermediate memory, and the second step would be to get rid of the internal TF forms and work exclusively with the ALF_3s. We could also use [43] to decrease the overall values of (r, K, C), but this should severely impact the signing time. Nevertheless, it is a very interesting research direction but one we did not pursue yet (maybe in a very near future!).

4.2 Further precomputations

$\mathsf{DVA}\text{-PC}_0$ essentially transforms the ALF_3s into a digest that is more palatable for the challenge generation (a bit-string of lower size), and doing so in a parallel manner that allows to reduce the intermediate memory. However, note that this operation does not need the message. In fact, we can then precompute those hashes *before* getting a message to sign. Thus, in this part we consider a scenario not considered the NIST framework, or rather not considered in the initial ALTEQ submission: a distinction between, an offline mode and an online mode, which allows us to perform precomputations offline. The topic of precomputations to accelerating schemes is not new [15][19]. This could be useful for a wide-range of applications, where the heavy part of the signatures can be computed in advance (namely, the group actions), and then we can just wait for a message to continue the rest of the signature process. Note that the precomputations we propose in this paper do not affect the verifications, thus while there should be some impact on asynchronous protocols such as TLS, the bottleneck of verification speed cannot be circumvented by this technique alone, although within certain conditions, verification times *could be* side-stepped[20]. However, if the goal is to send an *expected* message quickly on obtaining the message information, then this technique can greatly reduce the latency between obtaining

[19] Thomas Plantard pointed out that MAYO [12] also considers precomputations and separated an offline expansion part [11] from their signing and verification process. Thus, our approach is not marginal among in PQC. We are at least 2!

[20] If we sign a message for step k of some protocol, but detect an invalid signature received in a previous step: we could abort the protocol before step $k+1$, assuming step k did not send anything compromising. This requires step k to be safe to perform regardless of what was previously received. Because we are not familiar with advanced constructions, this could be a very terrible idea.

a message and signing it, assuming the necessary extra resources are available. For example:

- When we can start precomputing the moment a message is being written i.e any situation where the sender of the message sign his own message (like a signed letter/package). This can be done in parallel or even before the preparation of the message: for example while writing a lengthy e-mail we wish to sign to some forum (say [pqc-forum]), we give plenty of time for the precomputations to be executed in the background, which in turn decreases the delay for the message to actually be signed and sent[21].
- If the message source is not the sender itself but is expected, a natural application arises whenever the message source is from a very far distant location (or a very laggy one): assuming extremely fast communication speed, let's say light speed (299792458 m/s), then an information sent from Paris to Tokyo takes more than 32 milliseconds, and more than 56 milliseconds from Paris to Sydney. For a lot of cryptographic schemes, this is the time taken by a large amount of signatures. In such use cases, precomputations makes a lot of sense especially since superluminal communication is theoretically impossible so far[22], and from personal experience we believe it is hard to have less than 300 ms of ping between Sydney and Paris[23].

To improve on the efficiency of the precomputations, we make full use of our new DVA-PC_0 construction.

The idea here is simple. The self-feeding hash challenge used in the verification and the signature in DVA-PC_0 does not make use of the ALF_3, but only of hashes. Thus, the very natural idea is to compute the hashes in advance, then wait for the message. Unlike ALF_3s, hashes are easier to store: we are not completely certain of the size the hashes need to be for security, but arbitrarily we set to the standard size of 2λ-bits for λ-bits of security. Note that in our case, we have *one precomputation per signature*. This is different from other forms of precomputations, that are done once per key pair for signing [22,12], or once per key and verifier, using public information only, for verifying [5]. And unlike [5], the precomputation does not compress internal long term storage, but only expands it (although temporarily), thus this suggestion is only applicable in application cases *where we have memory to spare*. The closest analogy we know of is probably the precomputation of Gaussian sampling [37], which also make use of an online/offline phase.

It is clear that if everything is done offline and assuming forgers only use and see the online data, then there is no *obvious* loss of security as no external

[21] For example, appending a PGP signature at the end. It also seems like MLS [7] also have a PrivateMessage component where the sender signs their own message, but we are not familiar with it.

[22] As far as we know quantum communications are still sublight speed.

[23] Over several years, we conducted lengthy and painful experimentations on particular "real-time" multi-users applications.

information is ever leaked or requested. However, if we consider offline attackers that have access to intermediate data, intermediate computations, etc... then there is a massive risk of permitting a "data thief" to sign messages without the secret key, although when and where this particular setting comes in real-life is not very clear to us: if an attacker can steal offline data, he might as well directly steal the secret key, so a particular attack model should be required to properly analyse this angle. Stealing the secret key offline would affect every other scheme after all. Currently, we are not sure which attack model to consider[24], and for simplicity it might be better to leave this for further work.

First level of precomputation The first basic transformation is to just save the information we need for the "self-feeding" hash. This lead to the split of the signature process into two parts, namely $\mathsf{PreSign1}_{\mathsf{DVA}}$ and $\mathsf{FinishSign1}_{\mathsf{DVA}}$, presented in figure 9, that we name $\mathsf{DVA\text{-}PC}_1$ (Precomputation 1), that omit failure management (that we discuss shortly later), or the salt (we do not make a clear choice between larger seeds and using a salt).

```
$\mathsf{PreSign1}_{\mathsf{DVA}}(\mathsf{pk})$
$\phi_0 \leftarrow \mathsf{expATF}(\delta_0)$
$\beta \leftarrow_R \{0,1\}^{\lambda},\ \mathtt{salt} \leftarrow_R \{0,1\}^{2\lambda}$
$\{\mathsf{s}_i\}_{[r]} \leftarrow \mathsf{expSeeds}(\beta, r)$
$\{B_i^{col}\}_{[r]} \leftarrow \mathsf{expCols}(\{\mathsf{s}_i || \mathtt{salt} || i\}_{[r]})$
$\{\psi_i\}_{[r]} \leftarrow \mathsf{ActATF}(\{\phi_0, B_i^{col}\}_{[r]})$
$\{h_i\}_{[r]} \leftarrow \{\mathsf{H}(\psi_i)\}_{[r]}$
$\textbf{return}\ \mathsf{PreSgn} = (\{\mathsf{s}_i, h_i\}_{[r]}, \mathtt{salt})$
```

```
$\mathsf{FinishSign1}_{\mathsf{DVA}}(\mathsf{sk}, \mathsf{PreSgn}, M)$
$\{\delta_i\}_{[\![0,C]\!]} \leftarrow \mathsf{expSeeds}(\mathsf{sk}, C+1)$
$\mathsf{Ch} \leftarrow \mathsf{H}(\mathsf{H}(M) || \{h_i\}_{[r]})$
$\{c_i\}_{[r]} \leftarrow \mathsf{expCha}(\mathsf{Ch})$
$\mathsf{S} \leftarrow \{\},\ \mathsf{I}_{\Delta} \leftarrow \{\}$
$\textbf{for}\ i \in [r]\ \textbf{do}$
    $\textbf{if}\ c_i = 0\ \textbf{then}\ \mathsf{App}(\mathsf{S}, \mathsf{s}_i)$
    $\textbf{else}\ \mathsf{App}(\mathsf{I}_{\Delta}, i)$
$\{B_i^{col}\}_{\mathsf{I}_{\Delta}} \leftarrow \mathsf{expCols}(\{\mathsf{s}_i || \mathtt{salt} || i\}_{\mathsf{I}_{\Delta}})$
$\{\Delta_{c_i}^{col}\}_{\mathsf{I}_{\Delta}} \leftarrow \mathsf{expCols}(\{\delta_{c_i}\}_{\mathsf{I}_{\Delta}})$
$\{D_i\}_{[K]} \leftarrow \mathsf{ColMul}(\{\Delta_{c_i}^{col}, B_i^{col}\}_{\mathsf{I}_{\Delta}})$
$\{D_i^{col}\}_{[K]} \leftarrow \mathsf{ColDec}(\{D_i\}_{[K]})$
$\mathsf{Sig} = (\mathsf{Ch}, \mathtt{salt}, \mathsf{S}, \{D_i^{col}\}_{[K]})$
$\textbf{return}\ \mathsf{Sig}$
```

Fig. 9: $\mathsf{DVA\text{-}PC}_1$: light precomputations

[24] The random oracle model does not answer our worries: we are concerned about vulnerabilities arising from *large secret temporary data* such as bypassing `ASLR` and other shenanigans we are honestly not familiar with. We could go on a tangent on models vs reality, but ultimately *any* way to measure the threat posed by *large temporary secret sizes* could be better than none.

The current form of the precomputation data PreSgn is linear in r and λ per signature, making it very light or heavy depending upon the size of r: we believe it is actually shorter than a single ALF$_3$ ($\binom{n}{3} \times$ 32-bits) for most parameters sets in ALTEQ, and very likely to be smaller than most cryptographically usable d-tensors ($\binom{n}{d} \times \log_2(q)$-bits). The columns are expanded twice to save memory for PreSgn since expCols is fast (or at least was with `AES-NI`), but we can choose to change the shape of PreSgn to store $\{B_i^{col}\}_{[r]}$ instead of $\{\mathsf{s}_i\}_{[r]}$ for marginally faster signing but add a quadratic factor n^2 in memory space.

Structure of the precomputation data Interestingly, this variation only use the public key to perform precomputations, and not the secret key, which may have repercussions in later parts of the paper. In particular, PreSign1$_{\mathsf{DVA}}$ can be performed by *any* party, and its content can be used by *both* signatories *and* verifiers: this, and the combination of low-size data linear in λ and r, make it an ideal tool for future fancy cryptographic constructions which we do not tackle here. This paper is solely focused on improving/modifying ALTEQ performance, plus there is a major security issue to consider: the seed used for the K matrices in the signature should never be leaked (as mentioned previously, see [9]). Note that only the seeds s_i needs to be *securely stored* as in *"no external actor can read the data"*: the hashes h_i are publicly available to the future verifier anyway, thus even publicly announcing in advance the ALF$_3$s or their hashes should not lower the security.

Algebraic precomputation There is a tiny algebraic change that only marginally improve the performance of ALTEQ, but could noticeably improve the performance of FinishSign1$_{\mathsf{DVA}}$: changing the structure of the secret key, but without affecting the algebraic security. Currently, in ALTEQ, the secret key is generated as a column decomposition and its inverse computed and used for the setup. The secret key is then recomposed at every signature, multiplied by another column decomposition into a non-column decomposition to be decomposed again. Thus we propose:

– To generate full matrices in $\mathrm{GL}(n, q)$ as the secret key, *not in column form.* We expand directly into n^2 values, and then compute the inverse of the matrices to generate the public keys. Note, that if we cannot compute an inverse, it means the matrix is not invertible: we then discard and resample.
– Every time we wish to sign, expanding the stored seed (or seeds) of the secret matrices will guarantee invertible recomposed matrices without checking, since checking was done already at setup time.

Or, the simpler change but that requires more long-term memory for the secret key: recompose the secret matrices from their column form, and store them. We stress that this tiny change mostly affects FinishSign1$_{\mathsf{DVA}}$, which main operation is not in fact the matrix multiplication, but the column decomposition algorithm due to the high number of modular inversions required. In the original ALTEQ, the group action takes most of the time.

Second level of precomputation We can further accelerate the online signing phase, if polynomial storage space is not considered a main issue, with *much heavier* precomputations both in memory and time. While $\mathsf{PreSign1}_{\mathsf{DVA}}$ and $\mathsf{FinishSign1}_{\mathsf{DVA}}$ have roughly the same complexity as $\mathsf{Sign}_{\mathsf{DVA}}$, we can choose to precompute much larger data to reduce the signature procedure to be essentially *one of the fastest post-quantum signing procedure* if we exclude the cost of the offline phase. We say potentially because it depends upon certain choice of parameters, and frankly we do not know the details of *every signature scheme on the planet*[25], but the reasoning is essentially the following:

- There is not a lot of procedures that are faster than a hash.
- A lot of post-quantum signature procedures *requires* to hash an object.
- A lot of them require that hash *before* their heaviest computations can start. For example, hash-and-sign procedures like Falcon [22].
- In this option, once a message is received, we just compute *two* hashes, and then insert data at correct positions which is *in theory* faster than hashing.
- Once the precomputation is done, we do not even need sk!

The details of the procedures $\mathsf{PreSign2}_{\mathsf{DVA}}$ and $\mathsf{FinishSign2}_{\mathsf{DVA}}$ are presented in figure 10, and we name the overall transformation $\mathsf{DVA}\text{-}\mathrm{PC}_2$ (Precomputation 2). Unlike $\mathsf{DVA}\text{-}\mathrm{PC}_1$, $\mathsf{DVA}\text{-}\mathrm{PC}_2$ actually needs the secret key for its offline phase, but does not need it for its online phase.

Since this lies outside the framework suggested by the NIST, it is very possible that such a transformation exist for many other submissions and that we are far from being the most efficient one. Furthermore, while the theoretical speed of $\mathsf{FinishSign2}_{\mathsf{DVA}}$ is incontestable, the drawbacks that comes with $\mathsf{PreSign2}_{\mathsf{DVA}}$ are also incontestable:

1. In theory, the failure rate increases drastically at the precomputation level when calling ColDec. Experimentally, it is still low[26], but it might no longer be negligible depending of parameters (as reducing q). One option is to store a failure symbol in PreSgn during $\mathsf{PreSign2}_{\mathsf{DVA}}$, and take another sample PreSgn if the failure symbol is selected in $\mathsf{FinishSign2}_{\mathsf{DVA}}$, i.e we *hope* that the challenge generation does not pick a failure case during the online phase. This does slightly increase the complexity of $\mathsf{FinishSign2}_{\mathsf{DVA}}$, but we discuss failure management in the next part of this subsection: there might be more attractive options.
2. While $\mathsf{PreSign1}_{\mathsf{DVA}}$ produced PreSgn that are linear in λ and r only, the precomputation procedure $\mathsf{PreSign2}_{\mathsf{DVA}}$ produces PreSgn that are linear in r, K, C, λ, q and quadratic in n. It is still manageable because the memory required is not exponential, but it is still several orders of magnitude larger ($\mathsf{DVA}\text{-}\mathrm{PC}_2$ takes $Cn^2 \log_2(q)$ times more offline memory than $\mathsf{DVA}\text{-}\mathrm{PC}_1$). NIST recommends 2^{64} signatures per key: unless we dynamically recompute some samples to store while we sign (with two dedicated separated hardware/processes), it is unlikely that managing so many samples of this size have no impact on performance.

[25] We welcome every feedback.

[26] We did not see any on our old PQC2024 submission code

```
PreSign2_DVA(pk, sk)
1 : ${\delta_i}_{[\![0,C]\!]} \leftarrow \mathsf{expSeeds}(\mathsf{sk}, C+1)$
2 : $\phi_0 \leftarrow \mathsf{expATF}(\delta_0)$
3 : $\beta \leftarrow_R \{0,1\}^\lambda,\ \mathtt{salt} \leftarrow_R \{0,1\}^{2\lambda}$
4 : $\{\mathsf{s}_i\}_{[r]} \leftarrow \mathsf{expSeeds}(\beta, r)$
5 : $\{B_i^{col}\}_{[r]} \leftarrow \mathsf{expCols}(\{\mathsf{s}_i||\mathtt{salt}||i\}_{[r]})$
6 : $\{\psi_i\}_{[r]} \leftarrow \mathsf{ActATF}(\{\phi_0, B_i^{col}\}_{[r]})$
7 : $\{h_i\}_{[r]} \leftarrow \{\mathsf{H}(\psi_i)\}_{[r]}$
8 : $\{\varDelta_i^{col}\}_{[C]} \leftarrow \mathsf{expCols}(\{\delta_i\}_{[C]})$
9 : for $i \in [r]$ do
10 :    for $j \in [C]$ do
11 :       $D_{(i,j)} \leftarrow \mathsf{ColMul}(\varDelta_j^{col}, B_i^{col})$
12 :       $D_{(i,j)}^{col} \leftarrow \mathsf{ColDec}(D_{i,j})$
13 :    $\mathsf{Sample}_i \leftarrow \{\mathsf{s}_i, \{D_{(i,j)}^{col}\}_{[C]}, h_i\}$
14 : $\mathsf{PreSgn} \leftarrow (\{\mathsf{Sample}_i\}_{[r]}, \mathtt{salt})$
15 : return PreSgn
```

```
FinishSign2_DVA(PreSgn, M)
1 : $\mathsf{Ch} \leftarrow \mathsf{H}(\mathsf{H}(M)||\{h_i\}_{[r]})$
2 : $\{c_i\}_{[r]} \leftarrow \mathsf{expCha}(\mathsf{Ch})$
3 : $\mathsf{S} \leftarrow \{\},\ \mathsf{D} \leftarrow \{\}$
4 : for $i \in [r]$ do
5 :    if $c_i = 0$ then $\mathsf{App}(\mathsf{S}, \mathsf{s}_i)$
6 :    else $\mathsf{App}(\mathsf{D}, D_{(i,c_i)}^{col})$
7 : return $\mathsf{Sig} = (\mathsf{Ch}, \mathtt{salt}, \mathsf{S}, \mathsf{D})$
```

Fig. 10: DVA-PC$_2$: heavy precomputations

Precompute the precomputations The only difference between PreSign1$_{\mathsf{DVA}}$ and PreSign2$_{\mathsf{DVA}}$ is the presence of matrices that should not be leaked to the public, which is a major difference between the two versions. While we present PreSign2$_{\mathsf{DVA}}$ here as an independent entity of PreSign1$_{\mathsf{DVA}}$, it does not have to be. It is totally possible to use PreSign1$_{\mathsf{DVA}}$ as a precomputation of PreSign2$_{\mathsf{DVA}}$: the computation of the ALF$_d$ are independent of the computation of the matrix products, they just share a same seed which is also an output of PreSign1$_{\mathsf{DVA}}$. In particular, we can afford to have a larger amount of samples out of DVA-PC$_1$ and fewer samples of DVA-PC$_2$ and convert samples of DVA-PC$_1$ to samples of DVA-PC$_2$ when necessary: since the samples from DVA-PC$_2$ are much larger in memory, it might be more practical to store only a few, and have a larger bunch of DVA-PC$_1$ samples in the back ready to convert.

Dealing with failures, erasing failures The easiest way to deal with failures is to follow the original scheme, and use another set of samples PreSgn and discard the old set[27]. However, it might be in our interest to be able to effectively use *all* samples we generated and never (if possible) discard them. To avoid ALF$_3$

[27] We can still use the PreSgn batch for *another* message: nobody should be able to see the difference as Sign$_{\mathsf{DVA}}$ and FinishSign1$_{\mathsf{DVA}}$ have identical output structure, except for timing attacks since a swap of memory addresses may take some time. But even

collision attacks, PreSgn should never be reused when successfully used to sign, but in a case of failure, PreSgn can be recycled *if a salt or a trick like DVA-OTP is not used.* Without a salt or DVA-OTP (assume just larger seeds), we can recycle the following way: for the position that failed (or any random position $i \in [r]$), swap this sample with another one with the same index in another PreSgn batch *that has not been used yet.* Due to the matrix expanders using the round indexes r_{id} as part of their seeds, the permutations advertised in [43] are not usable unless we discard r_{id} (but we can keep `salt`). And if there is neither any `salt` nor r_{id}, anything can be swapped and the division into distinct PreSgn batches is not relevant. To be able to swap between batches we must obviously to have at least two batches PreSgn of samples precomputed. To give a mental image, use figure 11 and imagine the batches of PreSgn as horizontal bit strings stacked vertically, and see [43]'s swaps as *horizontal swaps* within the same PreSgn, and what we propose here as *vertical swaps* between two distinct PreSgn. If *all vertical swaps* fail (B^r possibilities for B batches PreSgn), then we have no choice but to resample.

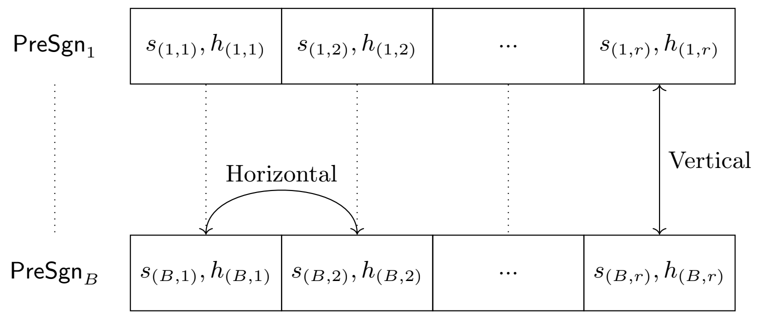

Fig. 11: DVA-PC's vertical swap (no `salt`) and [43]'s (no r_{id}) horizontal swap.

The other, expensive option to deal with failures is to make sure *no failure exists* during the online phase. This can be achieved in both DVA-PC$_1$ and DVA-PC$_2$ by performing the computation done in DVA-PC$_2$ *and reject every seed s* that expands into M such that there exists a choice of a secret matrix $\varDelta$ where $\mathsf{ColDec}(\varDelta M)$ fails. DVA-PC$_2$ already performs all possible combinations, thus it should not add too much complexity to its offline phase. DVA-PC$_1$ however, would take a massive toll to its offline performance if this solution is chosen. However, we can imagine a world where DVA-PC$_1$ is used with such an approach: this completely removes the need of a check in $\mathsf{FinishSign1}_{\mathsf{DVA}}$ and $\mathsf{FinishSign2}_{\mathsf{DVA}}$,

in the case of timing attacks (online, not offline), the information learned is really low, and we have yet to see a proper exploit.

ignore any permutation shenanigans and guarantees *algorithmic* constant-time in all cases[28].

4.3 Deeper into the rabbit hole: DVA-HA and DVA-PC-HA

We showed that replacing ALF$_3$s with their hashes for the challenge computation allows lower intermediate memory, and have lesser storage when using precomputations. Here we show here that by hashing further, a supplementary hash allows gaining further computational speed and *may* save even more memory for precomputations. This supplementary hash also concern the challenge computations: we propose to *hash all* the ALF$_3$s or their hashes into a single hash value. We name this option DVA-HA (*Hash All*), which is compatible with ALTEQ and DVA-PC$_{0/1/2}$. Since the cost of a hash is mainly decided by the size of the entry, adding an extra hashing step $\mathcal{H} = \mathsf{H}(\{h_i\}_{[r]})$ or $\mathcal{H} = \mathsf{H}(\{\psi_i\}_{[r]})$ to the computation of the challenge Ch, which facilitates the final challenge computation. In short, the final challenge computation for a message M is

- $\mathsf{Ch} = \mathsf{H}(\mathsf{H}(M)|\mathcal{H}) = \mathsf{H}(\mathsf{H}(M)|\mathsf{H}(\{h_i\}_{[r]}))$ for DVA-PC$_{0/1/2}$.
- $\mathsf{Ch} = \mathsf{H}(\mathsf{H}(M)|\mathcal{H}) = \mathsf{H}(\mathsf{H}(M)|\mathsf{H}(\{\psi_i\}_{[r]}))$ for ALTEQ.

and if the message M is constant size (in both cases), we can discard one hash and obtain $\mathsf{Ch} = \mathsf{H}(M|\mathcal{H})$ i.e a *single hash* for the online phase. We could also even fix the size of M (or $\mathsf{H}(M)$) and $\mathcal{H}$ to properly fit SHA256/384/512 and make use of the dedicated, already existing hardware intrinsics for SHA2 such as SHA-NI [25][29]. This accelerates even further the online signing phase of DVA-PC$_{1/2}$, but adds 2λ-bits of storage for PreSgn and *partly* disable the permutation techniques to deal with failures, since $\mathcal{H}$ changes for any bitwise change. We describe DVA-PC-HA as an example in algorithm 12.

Note, that in the case of DVA-PC$_{1/2}$, if we choose to ignore dealing with failures (by token swap), we could replace the $(2r\lambda)$-bits used for challenge computations (namely $\{h_i\}_{[r]}$), with simply the hash of all samples $\mathcal{H}$ of size (2λ)-bits. The only swap possible then in case of failure is to swap the whole PreSgn, i.e. swap two distinct $\mathcal{H}$ leaving less margins to deal with failures in matrix decompositions. Since we still need to store the seeds, which have total size $(r\lambda)$-bits, the interest of this technique is to divide by 3 the internal storage of PreSgn for DVA-PC$_1$ (from $(3r\lambda)$-bits to $((r+2)\lambda)$-bits), but has almost no impact in DVA-PC$_2$ for most parameters (the matrices represent the overwhelming majority of the memory for most parameters).

[28] We do not guarantee constant-time implementation-wise, as there could be cache misses, unpredictable memory access times, and so on *especially when the managed data is very large*, such as here in DVA-PC$_2$

[29] SHA512 intrinsics are recent, thus might not be common in most machines. Their documentation seems less than a year old: https://www.intel.com/content/www/us/en/developer/articles/release-notes/intrinsics-guide-release-notes.html.

```
PreSign_{DVA-HA}(sk)
        ⋮
15 :    $\mathcal{H} \leftarrow \mathsf{H}(\{h_i\}_{[r]})$
16 :    App(PreSgn, $\mathcal{H}$)
17 :    **return** PreSgn
```

```
FinishSign_{DVA-HA}(PreSgn, $M$)
 1 :    $\mathsf{Ch} \leftarrow \mathsf{H}(\mathsf{H}(M)||\mathcal{H})$
        ⋮
```

```
Vf_{DVA-PC-HA}(pk, $M$, Sig)
        ⋮
 9 :    $\mathcal{H} \leftarrow \mathsf{H}(\{h'_i\}_{[r]})$
10 :    $\mathsf{Ch}' \leftarrow \mathsf{H}(\mathsf{H}(M)||\mathcal{H})$
        ⋮
```

Fig. 12: DVA-PC-HA: extra hash (changes from DVA-PC$_{1/2}$)

Reducing storage size to exactly 3λ*-bits with DVA-PC*$_1$*-HA* We said earlier that DVA-PC-HA *may* reduce the storage cost even further. This is mostly true for DVA-PC$_1$ and the current ALTEQ implementation. In practice, the seeds used in ALTEQ are expanded from *one single seed*, i.e we use *one* seed of λ-bits to create r more seeds of λ-bits: the original *one* seed is never communicated. Since this is the case, we could store this *one source seed* instead of the r seeds, leading with $\mathcal{H}$ to a storage of only 3λ-bits per signature. If the salt is included in that seed expansion, we can maintain that 3λ-bits storage, otherwise this might rise to 5λ-bits with a salt of 2λ-bits. Note that if a seed of λ-bits was deemed too weak, we could double the source seed size and store 4λ-bits instead.

4.4 Accelerating expCha: using the combinatorial number system

With DVA-PC$_2$, the main online computation is the hash and the challenge generation. Currently, the challenge generation in ALTEQ is implemented as extracting the first 256-bits of a challenge hash and using them as a seed for an `AES-CTR`-based random bit generator and sample the values from this generator[30]. Reminder on the challenge shape:

1. Essentially, we are sampling K *distinct* numbers within $[r]$.
2. For each of those K positions, we sample randomly a value in $[C]$. Those values do not have to be distinct.

If we admit that the challenge hash cannot be manipulated by an attacker, then by considering the challenge hash as a random value x within $[2^{2\lambda}]$ we can just extract the K values in $[C]$ by considering x in base C and taking its first or last K digits: no need for relying on `AES-NI`. Then the job is to extract K distinct numbers within $[r]$ from $x - C^K$ and every combination must be equally probable

[30] This is for *all* parameters, any bit beyond the 256-th is ignored. It seems exploitable for an attacker, but we do not know how.

(as much as possible anyway). Thus, here comes our proposal: the combinatorial number system[31].

The combinatorial number system allows extracting from a large number N a strictly decreasing sequence $c_K > ... > c_1 \geq 0$ where

$$N = \binom{c_K}{K} + \cdots + \binom{c_2}{2} + \binom{c_1}{1}$$

In particular, the largest number in our case would (very naturally) be

$$N_{\max} = \binom{r-1}{K} + \cdots + \binom{r-K+1}{2} + \binom{r-K}{1} = \binom{r}{K} - 1$$

which leads to the combination $r > r-1 > \cdots > r-K$, with the lowest number being 0 represented by $K > K-1 > \cdots > 1$, thus mapping one-to-one every possibility and $[0, N_{\max}]$. We believe that extracting the choices among C from $(x \mod C^K)$ then the combination from $\left((x - C^K) \mod \binom{r}{K}\right)$ should be faster than an `AES-CTR`-based sampler, but we have yet to implement the algorithm and test. Plus, `AES-NI` is *really* fast. At least our suggestion could provide a gain for platforms where `AES-NI` is not available. Currently, we are thinking to implement the search of the combination in a one-or-two-dimensional array of size $r \times K$, where a basic greedy algorithm will diagonally traverse the array to choose the c_i. We also believe that a parameter C as a power of 2 will make extracting the K positions among C really fast by shift and mask operations.

4.5 DVA-PC into DVA-Tok: Adding a token to further exploit hashes

In this part we propose *another* failure management method, through exploiting the properties of hashes even further and sidegrade [43]'s other variation of ALTEQ. DVA-Tok however *increases* the signature size, although very marginally: in practice, the increase should be less than a dozen of bits. In a recently accepted paper, [43] proposed to introduce a rejection probability to the challenge generation to reject combinations of r ALF$_3$s, which remove the ability of the owner of the secret key to be able to sign any series of r ALF$_3$s, but allows to decrease the GMW-FS parameters (r, K, C) as a result to compensate the increase of the heuristic security introduced by the probability function. This probability function is computed at the same time as the challenge, by requesting the hash function to produce b extra bits, and then compare the number to some predetermined value and reject the combination of ALF$_3$s if the value is too high or

[31] This was mentioned by [31] who apparently quoted a work from the 19th century we could not verify, and was probably written in Italian.

accept if it is low enough (this could be reversed)[32].

The *practical* problem in [43], is that we have to either resample and recompute r group actions for each failure (which could range in the thousands in average to get *one* correct combination, depending on the parameters). To tackle this practical issue, [43] proposed to use permutations such as Heap's algorithm [26] to find valid combinations [43]. Problem: this is no longer a valid approach without significant changes to the current ALTEQ, since a round index r_{id} has been appended to the seed for matrix expansion to patch Beullens' seed collision attack [9]. Since we cannot permute horizontally samples within one batch without increasing the seed size (to patch Beullens' attack), we could use DVA-PC$_{1/2}$-HA or DVA-PC$_{1/2}$ to vertically swap several precomputed samples from distinct batches instead. However, since we could expect up to several thousands of vertical swaps *per signature* instead of an occasional vertical swap *once for every few thousands of signatures*, another idea to improve [43]'s non-constant signing time is highly desirable. Still, a slight change to ALTEQ can make this work practical, or at least with DVA-HA or DVA-PC-HA by re-using the reasoning used so far.

DVA-Tok's idea is to add an extra token t seen as a bit string of arbitrary size. Recall that [43]'s security assumption was, in short: for λ-bits security, just ensure the probability of a random trial has success chance $2^{-\lambda}$. Since the ALF$_3$s are only seen as bit strings in the challenge computation, extending that bit string by a value t controllable by the signatory should not drastically affect the security (we discuss this later). Let's suppose that Ch now also contains the permutation acceptance boolean with probability p. Thus:

- Initially, we use a challenge/validation computation with $\mathsf{Ch} = \mathsf{H}(\mathsf{H}(M)|\mathcal{H})$, or anything that was only ALF$_3$s and message M dependent.
- Instead, use $\mathsf{Ch} = \mathsf{H}(t|\mathcal{C})$ where $\mathcal{C} = \mathsf{H}(\mathsf{H}(M)|\mathcal{H})$, or $\mathcal{C} = \mathsf{H}(\mathsf{H}(M)|\{h_i\}_{[r]})$, or $\mathcal{C} = \mathsf{H}(\mathsf{H}(M)|\{\psi_i\}_{[r]})$, or even $\mathcal{C} = (\mathsf{H}(M)|\{\psi_i\}_{[r]})$, etc.
- The bit string t is independent from any other input of H, and can be chosen freely or enumerated. While non-constant time, this operation only deal with publicly available thus do not weaken security.

This extra hashing allows the signatory (and a forger, but the security model of [43] is unchanged) to resample rejected ALF$_3$s combinations by resampling new tokens t. The size of the token is up to the cryptographer: in fact, t with size 0 can be considered equal to the original ALTEQ scheme or any of the DVA. We do however advise making full use of intrinsics: have t get a size lower than 2λ, so that $\mathsf{Ch} = \mathsf{H}(t'|\mathcal{C})$ can be computed with hardware intrinsics specific to

[32] Note that in hindsight, those b extra bits are not necessary in practice. For a challenge value Ch larger than the necessary size λ-bits, we could reject deterministically a fraction p of all possible values of Ch as long as there are 2^{λ} non-rejected combinations left. Technically, this can be easily done in combination with the combinatorial number system we introduced since the hash value has size 2λ in the ALTEQ code.

SHA256/384/512 for example, where t' is t appended with zeroes or any other string easily computable to the verifier. Note that in practice, there is no reason to have t larger than the number of samples we are willing to try since t increases the signature by $\log_2(t)$, so this condition might always be verified. And at the same time, adding this extra bit token is also useful on its own to resample a new challenge in the case the factorization into column matrices fail, allowing for new valid challenges without recomputing group actions. In particular, we can use salt and DVA-OTP, r_{id} instead of larger seeds and not have to recompute or permute ALF_3, their hashes, batch of PreSgn and so on in case of a column decomposition failure.

Saving one hash using precomputations It is possible to set $\mathsf{Ch} = \mathsf{H}(\mathsf{H}(M)|\mathcal{C})$ with $\mathcal{C} = \mathsf{H}(t|\{h_i\}_{[r]})$ instead and precompute several $\mathcal{C}$ in advance for several token values t, but we do not propose it here. Assuming enough token are precomputed, we reduce the online signing speed by *only one single call* to a hash. Since there is a non-zero chance that all token lead to invalid challenges (or that not enough token were precomputed), we assume modifying t during the online phase is the most natural choice since the number of trials would theoretically not change much and remain exactly unpredictable per message[33].

DVA-Tok's security By adding a token into the protocol, we simplify resampling through the signing phase but also simplify the forgery attempts as a forger can attempt a new token instead of a new permutation. However, we believe that asymptotically we should not be too worried about this. The message M is a bit string goes through one or two more hashes than the token, or even none[34], before the final hash for the computation of the challenge, so both the message M and the token t are bit strings that go through similar operations at the same protocol step. In short: since the message and the token shares relatively close or identical structure links in the whole scheme, an attack through the token could also mean chosen ciphertext attacks in the original ALTEQ. If ALTEQ is secure through manipulation attacks of the message M, so is DVA-Tok.

4.6 Timing tests: TODO (but DVA-PC$_2$-HA should be more than 4/5 times faster than Dilithium)

Currently, we did not have the time yet to re-implement all our procedures in an acceptable way. A previous version as part of the PQC2024 submission on an old ALTEQ modification that is no longer relevant showcased times that were 4 faster than the level 3 of the AES-NI variant of Dilithium, and 5 times faster for their level 5 parameters. We also stated that comparing a precomputation variant with an implementation that does not have one was not fair, even though our

[33] SHA3, SHA2, or any chosen one-way function should not have malleable outputs

[34] Recall that ALTEQ's justification to hash the message was to facilitate the implementation for messages of flexible size. For constant-sized messages, this hash might not even be needed.

code was poorly optimized and lost 70% of performance in memory management (we will aim to improve that). Furthermore, we believe we should also compare ourselves to MAYO [12], in particular their timing with precomputations that came to our attention after PQC2024, notably in the first few minutes of the talk given for their update [11]. Their reported timings are also faster than Dilithium and make for a fairer comparison, although we have no idea yet if the level of optimization between their code and ours are equal, and we are not very familiar with their scheme either: pros and cons will have to be discussed, like memory sizes for example. We also apologize that due to reasons independent of our will, we cannot publicly release the old code of PQC2024, nor publicly provide a working repository at the moment[35]. We will however attempt to publicly release a better version as soon as we can, although we do not have a clear calendar for this. On a technical note, we would like to get our hands on a machine that has the `SHA512` intrinsics available, as this would greatly accelerate our modified challenge generation beyond the level 1 security.

5 DVA-DM: Dangerous Matrices for efficiency

5.1 Rationale

The biggest bottleneck in computation time is the group action, while the biggest bottleneck in signature size is the matrix size. To deal with the group action computation, we could craft either specialized ALF_d, or special matrices yielding faster computations: the latter seems more accessible for now. Intuitively, even if we wanted to compress the manipulated ALF_3s, we must make sure that the group action operation maintains a compressed form. Thus, it makes sense to work on the matrices first, then consider their links to the ALF_3s. In this section though, we merely just mention the matrices: link to an eventual ALF_3 form could be discussed later. Furthermore, since it is the biggest culprit in signature size, proposing matrices that carry a more compact structure in the signature could be interesting as well.

The research of specific matrix structures did not start with this paper: prior research was conducted on which matrix groups were leading to easier-to-solve ATFE instances, leading as far as possibly dropping an entire complexity class level [16]. In particular, according to [16], the orthogonal group and the unitary group are easier to solve than $\mathrm{GL}(n, q)$ in practice due to the presence of isomorphism invariants. Beullens' attack [10] also seems to rely on finding invariants. Although both orthogonal and unitary groups are still GI-hard (Graph Isomorphism) [16], the GI class of complexity was shown to be "not so hard" [4] (but not polynomial either).

In ALTEQ, the matrices mostly belong to $\mathrm{GL}(n, q)$ but are chosen *only* among those which admit a column decomposition without permutation. The aforemen-

[35] The reasons are not *purely* technical, but they are not permanent roadblocks.

tioned set is clearly not a group: hence why there is a failure test for the computation of the column decomposition. Thus following this train of thought, we search for structures that improve the algebraic computational efficiency with sacrificing as little as possible in security, or if we are sure we will lose "some" anyway, to gain as much efficiency as possible. We follow three trains of thoughts, and provide cryptanalytic targets as examples:

– Matrix structures solely decided on what we could do with available specific hardware accelerations, implementation-wise.
– Matrix structures solely bent on compressing the signature size through introducing some dependencies.
– Matrix structures solely focused on avoiding as many operations as possible from an algorithmic point-of-view.

Note that we have not tested any of the performance gains yet, but will do so in the future. Especially if any variations of those structures are shown to be secure: this would give extra motivation.

5.2 Implementation-wise only considerations

To research which form of algebraic structure could be interesting to use instead of $\mathrm{GL}(n, q)$, we first and foremost check which basic arithmetic operations are the most efficient on most computers. To see which operations could be efficient especially for vectorized instructions, our initial approach is both a mix of algebra and "lego blocks"[36] by looking at the available pool of operations at [1]. We have two reasons to use [1]: the first is Intel x64 architecture being targeted by the NIST in their signature calls (other targets were also encouraged, but not explicitly named as far as we know), the second is our shameful lack of knowledge concerning other targets[37]. While `AVX512` registers (the `zmm`) do not seem to be widely available[38], it does seem that the `AVX2` registers are (the `ymm`) and even beyond Intel processors, thus we focus on `AVX2` in particular. From our quick observations:

– Additions, subtractions, shifts, logical operations `or`, `xor`, `and`, `andnot` and comparison functions are available for all integers sizes of 8, 16, 32 and 64-bits in `AVX2` vectors. We can flip signs for all but the 64-bits size. All those operations seem to be close in performance. Multiplications however, are slower, not always consistently available and sometimes benefits from special operations, depending on the size.

[36] This is our personal opinion on the matter, and maybe why we think it is fun. More experienced implementers might feel differently, or hate Legos.

[37] We are *very* willing to learn though.

[38] For now, but who knows: `AVX512` can be efficiently simulated through some "double pumping" [2]. Thus, `AVX512` might become widespread in the future despite Intel's intent on their own creation. Native `AVX512` is rumoured to be present on one of AMD's upcoming chips Zen5, while Intel may be pushing for `AVX10` and `APX` instead [3]. We currently lack the competence to either predict or encourage a direction.

– For 64-bits integers, we can perform multiply-and-add for 52-bits integers, extracting either the low part or the high part of the 104-bits intermediate results (only with `AVX_IFMA`). Currently, we are not sure how `GCC` could detect those use cases for those instructions without forcing intrinsic calls.
– For 32-bits integers, we can multiply and either keep the low part, or the whole 64-bits results if we ignore half of the entry vectors.
– For 16-bits integers, we can multiply and keep the low parts, or the high parts. We believe the ability to use low and high parts of the multiplication results was key in the modified Montgomery algorithm used in Seiler's work [41], and instrumental in the efficiency of Dilithium [18]. We can also do a multiply-and-add combined with a horizontal sum that results into 32-bits values, which could be interesting to us: using extra shifts and modular reductions can leverage this instruction into a modular-multiply-and-add sequence (depending on the moduli size).
– For 8-bits integers, we can also do a multiply-and-add combined with a horizontal sum that results into 16-bits values. We can use this specific instruction to multiply integers into 16-bits values by filling half the vector with zeroes. There are also some forms absolute-difference operations for which we did not find a way to leverage yet.

For ALTEQ, the size of q forces us to 32-bits and 64-bits instructions[39] and currently the instruction `vpmuludq` generated by `GCC` is extensively used to produce 64-bits integers to be added, using 32-bits integers within half of the vector entries. Now that the moduli size has been reduced to 24 or 21-bits according to the latest poster [14], maybe using `vpmadd52luq` when `AVX_IFMA` is available could be worthwhile despite the documentation on the Intel website lacking information on the latency or the throughput (CPI) of that instruction.

Since the moduli size has decreased with the latest update [14], one idea is to simply use smaller entries for B (such as 8 or 11-bits integers) so we can multiply 8 32-bits integers (which are initially 24 or 21-bits) with one instruction and keeping the result within 32-bits, using `vpmulld` for example, despite its apparently "undefined" or very high latency. Another idea we propose is to use low weight integers entries, such that the multiplications can be replaced by one, two or three shift-then-add operations: each matrix entry can have its own shift value through `vpsllvd`. Thus, if we use the parameters n 18 and 27 as given in [14], multiplying through a single power of 2 (i.e just operating a shift) in each position already yield a number of possible matrices of $8^{18^2} \approx 2^{972}$ for the security level 1 and $11^{27^2} \approx 2^{2521}$ for security level 2 thus we *may* discard concerns about bruteforce attacks, however new algebraic attacks could see rise (thus using weight 2/3 integers instead or just one, or even adding signs). To keep a cryptanalytic target we keep just the simple shift matrix: it is interesting to see which new attacks could arise from targeting group actions computed from a matrix containing exclusively low powers of 2. The matrix B used algebraically

[39] Although 4-tensors could help push q below 16-bits, to allow us to use those sweet 16-bits instructions.

will be *internally* represented by a matrix S, where $S \in \{0, ..., 7\}^{n\times n}$ for security level 1 such that

$$B = \begin{pmatrix} 2^{S_{1,1}} & \dots & 2^{S_{1,n}} \\ \vdots & & \vdots \\ 2^{S_{n,1}} & \dots & 2^{S_{n,n}} \end{pmatrix}$$

Beyond the computational acceleration, this suggestion opens the way for specific arithmetic forms to be used. Forcing integers to be of certain values could remove the checks for zeroes in the column generations for example, and simplify the code. We could also craft set of matrices such that we can guarantee that a product admits a column decomposition without permutation, although we do not see a way to produce this yet. For now, we suggest to check first how weak those "shift matrices" actually are.

5.3 Adding some polynomial structure

The idea here is to use an algebraic structure that is inspired from what has been done in lattice-based cryptography. Basically, we can replace matrices with non-linked rows with algebraically linked rows, essentially having matrices being represented by polynomials instead (such as NTRU [27]). In the simplest form[40], this allows to replace $n \times n$ entries with just n entries. The product of the matrices is the matrix generated by the product of the polynomials, given that the algebraic link between the rows is the same.

Let us illustrate our statement with an example. Let $P_A, P_B \in \mathbb{F}_q[X]/\langle G\rangle$ where $G = X^5 - 1$ is the algebraic link between the rows and P_A, P_B are the polynomials associated to A, B respectively. Each row $\mathbf{a}_{i+1}$ is generated from the previous row $\mathbf{a}_i$ as ($P_{\mathbf{a}_{i+1}} = P_{\mathbf{a}_i} \times X \mod G$). Then, still for the sake of the example, set $P_A = X + 1$ and $P_B = X^2 + 1$, the matrices A and B are then

$$A = \begin{pmatrix} 1\,1\,0\,0\,0 \\ 0\,1\,1\,0\,0 \\ 0\,0\,1\,1\,0 \\ 0\,0\,0\,1\,1 \\ 1\,0\,0\,0\,1 \end{pmatrix},\ B = \begin{pmatrix} 1\,0\,1\,0\,0 \\ 0\,1\,0\,1\,0 \\ 0\,0\,1\,0\,1 \\ 1\,0\,0\,1\,0 \\ 0\,1\,0\,0\,1 \end{pmatrix},\ A \times B = \begin{pmatrix} 1\,1\,1\,1\,0 \\ 0\,1\,1\,1\,1 \\ 1\,0\,1\,1\,1 \\ 1\,1\,0\,1\,1 \\ 1\,1\,1\,0\,1 \end{pmatrix}$$

and one can check that indeed, $P_A \times P_B \mod G = P_{AB} = X^4 + X^3 + X^2 + 1$. Since the matrices given in the signature are typically just a product of matrices, then assuming the matrices generated by a seed for the signature and the those of the secret keys share the same algebraic link, then we can give the matrices as just a polynomial $P_{\text{signature}} = P_{\Delta_i} \times P_B$, reducing $n^2 \log_2(q)$-bits object into a $n \log_2(q)$-bits object. However, we did not find a way to leverage this structure to accelerate the computations of the group actions, and the column decomposition might still have to be computed in that regard: thus the speed advantage

[40] We are not going to talk about ideal/module lattices and use as little background as possible. For lattice-people (and beyond), this subsection may be trivial.

is really unclear to us (maybe aside from matrix multiplication, using NTTs, Karatsuba [29] or a mix of both [32]), and only the signature size may benefit from this structure.

Note that this was somehow proposed in our initial submission to PQC 2024, but we also recommended against it because of our fear of an incredibly weak structure: any attack involving polynomial system solving will now have only n variables to guess instead of n^2, plus adding potential dependencies into the ALF_3 coefficient equations that did not previously exist (especially due to the algebraic links between the rows). However, the submission to PQC 2024 was *before* the updated algebraic attack of [38] which may lead to a significant increase of n. Hence, a transformation from n^2 to n is now more attractive and we are no longer discarding this idea as quickly. Furthermore, since we are providing cryptanalysis targets in this paper, we might as well set an easy target to encourage the understanding of such objects. First and foremost, we would like to know how much n or q have to be increased to reach a certain amount of security, or if the attacks are in fact polynomial time and thus this approach is infeasible. And how would those attacks carry-over on d-tensors?

Let n', q' be the parameters necessary to reach a certain security level for the polynomial structure we just described, and n, q the traditional parameters in $\mathrm{GL}(n, q)$ for the same security level. It is very likely that $n' \geq n$ and/or $q' \geq q$, but as long as $n' \log_2(q') < n^2 \log_2(q)$, then there is a gain in the signature size. Intuitively we believe the structure could be broken quite easily, but our interest in this structure does not stop there. In the likely case there is no practical parameters in this "one-block" structure, why not take inspiration of the special matrices used in the two lattice-based NIST finalists, Dilithium [18] and Falcon [22]? We could cut it into a 2×2 matrix where each block of size $n/2 \times n/2$ is generated by a polynomial, or split even further until the matrix is composed of $n/2 \times n/2$ blocks of size 2×2, which would still divide the signature size by 2. Either no cut lead to a gain in signature size, either the proper cut needs to be found: see the blocks A below from $n \times n$ to 2×2:

$$\left(A\right),\ \left(\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline A_{2,1} & A_{2,2}\end{array}\right),\ \ldots,\ \left(\begin{array}{c|c|c|c|c} A_{1,1} & A_{1,2} & \cdots & A_{1,(n/2)-1} & A_{1,n/2} \\ \hline A_{2,1} & \ddots & & & A_{2,n/2} \\ \hline \vdots & & \ddots & & \vdots \\ \hline A_{n/2-1,1} & & & \ddots & \vdots \\ \hline A_{n/2,1} & A_{n/2,2} & \cdots & A_{n/2,(n/2)-1} & A_{n/2,n/2}\end{array}\right)$$

Could we find heuristically secure trade-offs or is this doomed to fail? And since we have less restrictions (no "short vectors" consideration for example, so no need to have the "algebraic link" G to be a small sparse polynomial) and smaller matrices, maybe there are openings for more creativity.

5.4 Removing computations with more zeroes

In the original rejected paper in PQC 2024, we proposed a structure to avoid as many computations as possible by introducing flexible zeroes positions that would avoid several multiply-and-add and modular reductions. As the reviewers pointed out, there was a miscalculation in the number of permutations but as the structure was scalable anyway the reasoning is still sound. Let us then present what we called "crossed-boxes" matrices, inspired by immigration questionnaire forms. The crossed-boxes matrices we proposed have the following form:

$$\forall P_1, P_2 \in S_n, M \in \mathit{\Gamma}_k, \text{ we have } B_i = P_1 M P_2 \in \mathcal{B}$$

where S_n is the set of permutation matrices, and $\mathit{\Gamma}_k$ is a scalable structure with k ranging from 1 to $n-2$ that we represent in the following form for $k=1$ in column decomposition:

$$M^{col} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & \cdots & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{1,2} & 0 & \cdots & \cdots & 0 & a_{2,n} \\ \vdots & 0 & \ddots & 0 & 0 & \vdots & \vdots \\ \vdots & \vdots & 0 & a_{i,i} & 0 & \vdots & \vdots \\ \vdots & \vdots & 0 & 0 & \ddots & 0 & \vdots \\ a_{n-1,1} & 0 & \cdots & \cdots & 0 & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & \cdots & \cdots & \cdots & a_{n,n-1} & a_{n,n} \end{pmatrix} \in \mathsf{S}_0$$

i.e, a matrix M^{col} with a column decomposition where only the following elements are non-zero: the first and last element of each row/column, and the diagonal values. Preliminary experimental computations shows that while such M^{col} are very sparse, the associated $M \in GL(n,q)$ are *completely* dense, and so are M^{-1} and their column decomposition $(M^{-1})^{col}$. Generalizing the structure give us $\mathit{\Gamma}_k$ the set of invertible matrices with column decompositions A^{col} such that the "contour" is dense and the "non-contour" has at most k non-zero elements per row and column. Obviously $\mathit{\Gamma}_1$ is the most "empty" one since diagonal coefficients need to be non-zero, but in case of security issues we can "slide" k until we reach the full representation with $k = n-2$ which are most of the ALTEQ matrices.

Note that a reviewer pointed that each column of $(M^{-1})^{col}$ is indeed linked to M^{col}. In particular, the same reviewer wrote that those matrices can be written as $A+R+C+Q$, where A is a rank 1 matrix, R is a matrix with one row, C is a matrix with one column, and Q is a monomial matrix i.e. a product of a diagonal and permutation matrix, at least for $\mathit{\Gamma}_1$. As stressed by the reviewer, this *does* seem exploitable by an attacker and we completely agree with the sentiment, but we are not sure how to attack this structure either. Exploiting the zeroes seems the way to go: after all, since $\mathit{\Gamma}_{n-2}$ is roughly equivalent to the matrices used in ALTEQ, $\mathit{\Gamma}_1$'s sparsity should be an obvious attack angle. The permutations make it hard to predict accurately where the zeroes are going to be. While there is a good chance to guess *one* of the zeroes (since most coefficients are zeroes), how many zeroes can we guess until one guess is likely to be false and thus wasting computing resources on an unsolvable algebraic system? Would it be possible to identify zero positions in polynomial time, leading to a full break?

The idea of the "cross-boxes" had their inspiration from filling forms, but the core reasoning is predictable zeroes for the signer, a staple idea that we took from UOV [30] schemes: knowing zeroes in a certain "secret transformation" lead to an easy system-solving, which could not be done with "random transformations". In the "cross-boxes" case, the transformation is only rows and columns permutations of the matrix: the zeroes are still there in plain *matrix* view, but the matrix is hidden as a solution to a system of multivariate polynomials. There could be more interesting, computationally efficient tranformations, that hide more securely the zeroes and still provide a computational gain: maybe exploiting DVA-GG and considering the product of multiple sparse structures (as ALTEQ's n column matrices, see figure 2).

Another problem come from the implementation of this "cross-box" structure, or all other permuted-sparse structures: the permutations themselves. While there is no field operations, in theory this should be a fast operation *sequentially.* In parallel (vectorization, etc...) this is another story. We do not know of a way to produce random permutations in a vectorized manner efficiently, and ALTEQ's current efficiency relies strongly on interwoven data. Shuffle operations *are* available [1], but mostly concerns elements within the same vector: not only you need several vectors to cover just one ALF_3 (thus we also need blending strategies), but this would also require the ALF_3 to be unvectorized. We stress this does not mean we automatically give up on this subject: how many distinct uniformly distributed permutations can we cover with a limited set of instructions, and if they are easily distinguishable. In particular, we might have to study shuffle groups and more obscure combinatorial structures. Or, a more radical solution, target hardware that does not depend on vectorization and have extremely fast random data swaps[41]. Challenging maybe, but interesting.

6 Open questions

We presented some variations of ALTEQ (DVA) to improve computational efficiency in different scenarios and explore new applications. We took risks in heuristic security, by stepping out of the usual constructions that were used in the conception of ALTEQ, while discussing the level of risk regarding the known attacks. Thus the following questions naturally come, that likely needs further work:

– We introduced DVA-GG as a generic view of the group actions operated in ALTEQ, leading to a potential countermeasure against Beullens' seed collision

[41] Not sure if that even exists?

attack. We also presented some example use cases of the intermediate graph points. So far those usages can also be realized with non-primitive related modifications, namely with successive hashes and/or expansions. Are there more creative way to use those intermediate points, especially in a way that would be specific to the primitive and cannot be replicated by hashes or expansions?

– DVA-PC$_{0/1/2}$ improves the efficiency of the results to be communicated if we delegate the computation of the group actions. We discussed the advantage of precomputations in asynchronous protocols or long distance communications, but that only concerns devices that can afford them. Could we construct further variations to DVA-PC that would suit less powerful devices? For example, fitting targets like the ones used in [28] may already be challenging for DVA-PC$_{1/2}$, let alone ALTEQ. What about smartphones[42]? Another use case of using hashes is to limit communications with precomputation devices. But how can we securely delegate the computations of the group actions, if the only return value we get is a hash? How realistic is a third-party precomputation device?

– We presented some matrix structures to either reduce the signature size or improve the overall speed of the signature and verification processes. The practical security is currently unknown for those matrices, and we would like to have a general study of those objects to see if there is some relevant instantiation that could make them secure. Or better, if there is some security reduction hidden somewhere? What would be interesting is to find another matrix structure that we could bind to the ALF$_3$ structure, or ALF$_d$ structure: this could be the only way to provide both compressed *and* faster objects.

Acknowledgments. All the ALTEQ team, who introduced the GMW-FS framework and the ATFE problem to us. Kazuhide Fukushima, Yuto Nakano and other members of KDDI Research Inc for introducing “real-life” stuff to us or other communities like the IETF. Saarinen and Beullens, simply because this work was inspired by their comments on [pqc-forum].

Disclosure of Interests. All members of the ALTEQ team, and by extension, people working in the same companies/institutes, such as KDDI Research Inc, Saarland University, SandboxAQ, Nokia Bell Labs, Birmingham University, the University of Wollongong and the University of Technology Sydney.

References

1. Intel® Intrinsics Guide , https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html
2. 4TH GEN AMD EPYC PROCESSOR ARCHITECTURE, https://www.amd.com/system/files/documents/4th-gen-epyc-processor-architecture-white-paper.pdf

[42] For example, https://developer.android.com/privacy-and-security/keystore does not seem to define a hard limit. Thus this could be a hardware problem only.

3. Introducing Intel® Advanced Performance Extensions (Intel® APX), https://www.intel.com/content/www/us/en/developer/articles/technical/advanced-performance-extensions-apx.html
4. Babai, L.: Graph isomorphism in quasipolynomial time. In: Proceedings of the forty-eighth annual ACM symposium on Theory of Computing. pp. 684–697 (2016)
5. Bajard, J.C., Fukushima, K., Plantard, T., Sipasseuth, A.: Fast verification and public key storage optimization for unstructured lattice-based signatures. Journal of Cryptographic Engineering pp. 1–16 (2023)
6. Baldi, M., Barenghi, A., Beckwith, L., Biasse, J.F., Esser, A., Gaj, K., Mohajerani, K., Pelosi, G., Persichetti, E., Saarinen, M.J., Santini, P., Wallace, R.: LESS (Linear Equivalence Signature Scheme) (2023), https://www.less-project.com/
7. Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., Cohn-Gordon, K.: The Messaging Layer Security (MLS) Protocol. Tech. rep., Internet Engineering Task Force (2023), https://datatracker.ietf.org/doc/html/rfc9420
8. Beullens, W.: Battle report (first 30 hours, add'l sigs round 1) (2023), https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/5JMFqozi1Bc/m/qnWnsAtxBQAJ
9. Beullens, W.: Trivial multi-key attacks + attack on alteq (2024), https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/tjhrrmv837w/m/sjxHooYgBAAJ
10. Beullens, W.: Graph-theoretic algorithms for the alternating trilinear form equivalence problem. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023. pp. 101–126. Springer Nature Switzerland, Cham (2023)
11. Beullens, W., Campos, F., Celi, S., Hess, B., Kannwischer, M.J.: Nibbling MAYO: Optimized implementations for AVX2 and Cortex-M4 (2024), https://csrc.nist.gov/Events/2024/fifth-pqc-standardization-conference
12. Beullens, W., Campos, F., Celi, S., Hess, B., Kannwischer, M.J.: MAYO: Practical Post-Quantum Signatures from Oil-and-Vinegar Maps (2023), https://pqmayo.org/
13. Bläser, M., Duong, D.H., Narayanan, A.K., Plantard, T., Qiao, Y., Sipasseuth, A., Tang, G.: The alteq signature scheme: Algorithm specifications and supporting documentation. NIST PQC Submission (2023), https://pqcalteq.github.io/ALTEQ_spec_2023.09.18.pdf
14. Bläser, Markus and Duong, Dung Hoang and Narayanan, Anand Kumar and Plantard, Thomas and Qiao, Youming and Sipasseuth, Arnaud and Tang, Gang: ALTEQ : ALternating Trilinear form EQuivalence, poster for the 5th NIST PQC Standardization Conference, online at https://pqcalteq.github.io/alteqposter.pdf
15. Boyko, V., Peinado, M., Venkatesan, R.: Speeding up discrete log and factoring based schemes via precomputations. In: Nyberg, K. (ed.) Advances in Cryptology — EUROCRYPT'98. pp. 221–235. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
16. Chen, Z., Grochow, J.A., Qiao, Y., Tang, G., Zhang, C.: On the Complexity of Isomorphism Problems for Tensors, Groups, and Polynomials III: Actions by Classical Groups. In: 15th Innovations in Theoretical Computer Science Conference (ITCS 2024). Schloss-Dagstuhl-Leibniz Zentrum für Informatik (2024)
17. Chou, T., Niederhagen, R., Persichetti, E., Ran, L., Randrianarisoa, T.H., Reijnders, K., Samardjiska, S., Trimoska, M.: MEDS: Matrix Equivalence Digital Signature (2023), https://www.meds-pqc.org/
18. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium algorithm specifications and supporting documentation. Round-2 submission to the NIST PQC project **35** (2021)

19. Ducas, L., Postlethwaite, E.W., Pulles, L.N., van Woerden, W.: Hawk: Module lip makes lattice signatures fast, compact and simple. In: ASIACRYPT 2022. Springer-Verlag (2022)
20. Espitau, T., Niot, G., Sun, C., Tibouchi, M.: Squirrels: Square unstructured integer euclidean lattice signature (2023), `https://squirrels-pqc.org`
21. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the theory and application of cryptographic techniques. pp. 186–194. Springer (1986)
22. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Fast-fourier lattice-based compact signatures over NTRU (2018), `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-2/submissions/Falcon-Round2.zip`, NIST PQC
23. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. Journal of the ACM (JACM) **38**(3), 690–728 (1991)
24. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. Journal of the ACM (JACM) **62**(4), 1–64 (2015)
25. Gulley, S., Gopal, V., Yap, K., Feghali, W., Guilford, J., Wolrich, G.: New Instructions Supporting the Secure Hash Algorithm on Intel Architecture Processors. Tech. rep. (2013), `https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sha-extensions.html`
26. Heap, B.: Permutations by interchanges. The Computer Journal **6**(3), 293–298 (1963)
27. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A ring-based public key cryptosystem. In: Algorithmic number theory, pp. 267–288. Springer (1998)
28. Kannwischer, M.: pqm4: Benchmarking NIST Additional Post-Quantum Signature Schemes on Microcontrollers (2024), `https://csrc.nist.gov/Presentations/2024/pqm4-benchmarking-additional-pq-signature-schemes`
29. Karatsuba, A.: Multiplication of multidigit numbers on automata. In: Soviet physics doklady. vol. 7, pp. 595–596 (1963)
30. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced oil and vinegar signature schemes. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 206–222. Springer (1999)
31. Knuth, D.E.: The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions. Addison-Wesley Professional (2005)
32. Moenck, R.T.: Practical fast polynomial multiplication. In: Proceedings of the third ACM symposium on Symbolic and algebraic computation. pp. 136–148. ACM (1976)
33. NIST: Post-quantum cryptography standardization (2018), `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography`
34. NIST: Post-Quantum Cryptography Standardization (2022), `https://csrc.nist.gov/news/2022/pqc-candidates-to-be-standardized-and-round-4`
35. NIST: Post-Quantum Cryptography Standardization (2022), `https://www.nist.gov/news-events/news/2022/07/pqc-standardization-process-announcing-four-candidates-be-standardized-plus`
36. NIST: Post-Quantum Cryptography Standardization (2023), `https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures`
37. Peikert, C.: An efficient and parallel gaussian sampler for lattices. In: Annual Cryptology Conference. pp. 80–97. Springer (2010)

38. Ran, L., Samardjiska, S., Trimoska, M.: Algebraic algorithm for the alternating trilinear form equivalence problem. In: Code-Based Cryptography: 11th International Workshop, CBCrypto 2023, Lyon, France, April 22–23, 2023, Revised Selected Papers. p. 84–103. Springer-Verlag, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-3-031-46495-9_5
39. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2) (1978)
40. Saarinen, M.J.O.: Official comment: Alteq (2023), https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/-LCPCJCyLlc/
41. Seiler, G.: Faster AVX2 optimized NTT multiplication for ring-LWE lattice cryptography. Cryptology ePrint Archive, Report 2018/039 (2018), https://eprint.iacr.org/2018/039
42. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing **26**(5), 1484–1509 (1997)
43. Sipasseuth, A.: Faster verifications and smaller signatures: Trade-offs for **ALTEQ** using rejections (2024), to appear in ACISP 2024
44. Takagi, N., Yoshiki, J., Takagi, K.: A fast algorithm for multiplicative inversion in GF(2^m) using normal basis. IEEE Transactions on Computers **50**(5), 394–398 (2001). https://doi.org/10.1109/12.926155
45. Tang, G., Duong, D.H., Joux, A., Plantard, T., Qiao, Y., Susilo, W.: Practical post-quantum signature schemes from isomorphism problems of trilinear forms. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022. pp. 582–612. Springer International Publishing, Cham (2022)