Received -; | Revised -; | Accepted -

DOI: xxx/xxxx

RESEARCH ARTICLE

Threshold Secret Sharing with Geometric Algebra[†]

David W. H. A. da Silva | Luke Harmon | Gaetan Delavignette

[1]Research & Development, Algemetric, LLC, Colorado, USA

Correspondence
*David W. H. A. da Silva. Email: dsilva@algemetric.com

Present Address
5555 Tech Center Dr, Colorado Springs, Colorado, 80919

Summary

As establishing a foundation for a new line of investigations on threshold secret sharing schemes with geometric algebra (GA), we propose a variation of a well-known threshold secret sharing scheme introduced by Adi Shamir in 1979, a cryptographic solution that allows a secret input to be divided into multiple random shares which are then sent, each one, to distinct parties. The computation of these shares is done so that there are proper subsets of these shares that allow reconstructing the secret input using polynomial interpolation. To reconstruct the secret input, Shamir's scheme requires a minimum number of shares, smaller than the total number of shares, referred to as a threshold. Any number of shares smaller than the threshold reveals nothing about the input secret. The random shares are generated such that each party can perform computations, generating a new set of shares that, when reconstructed, are equivalent to performing those exact computations directly on the secret input data. Our variant replaces the algebra in which the original secrets lie from integers to GA while preserving fundamental properties in Shamir's scheme, such as perfect secrecy and *idealness* (both secret and random shares are members of the same space). As a direct result, any application in GA dealing with multivectors can immediately add threshold security using our scheme. Non-GA applications can also benefit from our results by using multivectors as a vessel for sharing multiple secrets at once.

KEYWORDS:
geometric algebra, threshold secret sharing, security, privacy, multiparty computation

1 | INTRODUCTION

In Information Security, the CIA triad refers to Confidentiality, Integrity, and Availability, fundamental principles that must be observed in any solution aiming to achieve the goals associated with security and privacy. These principles are mostly addressed by cryptographic tools. Although all principles are equally important and play complimentary roles with respect to each other, confidentiality is probably the most ancient and easy to understand need in security and privacy, even for those that are not familiar with the subject. The fact is everyone understands the need for confidentiality, no matter what precisely is being protected. Encryption is commonly the go-to solution for ensuring confidentiality, that is, preventing the unauthorized disclosure of data. Other tools can also contribute for ensuring confidentiality such as mechanisms of identity and access management (IAM) including identity, authentication, authorization, and accounting.

[†]This research is supported by Algemetric, LLC.

[0]**Abbreviations:** PET, privacy-enhancing technology, TSS, threshold secret sharing; MPC, multiparty computation; GA, geometric algebra

However, as society advances, its needs become more sophisticated. In 1978, as an example, Rivest et al. remark [1] that encryption is, in a sense, limited. That is, once data is encrypted, one can only store it and retrieve it. In order to use that data for any meaningful computation one would need to decrypt it first. Rivest et al. then observe the existence of encryption functions which allow computations over encrypted data for many sets with interesting operations. Such encryption functions were then called *privacy homomorphisms*. Later on, several authors proposed several practical demonstrations of specific families of computations on encrypted data [2,3,4,5]. In 2009, the first formal evidence that any function can actually be computed via homomorphic encryption was introduced [6] and since then many other contributions are pushing forward the field of homomorphic computation over encrypted data [6,7,8,9,10,11].

The National Institute of Standards and Technology (NIST) through its Cryptographic Technology Group (CTG) in the Computer Security Division (CSD) is promoting the progress and adoption of emerging technologies in the area of privacy enhancing cryptography (PEC) [12] as enabling tools for achieving privacy goals. Among other aspects, achieving privacy goals involves computing over encrypted data and allowing multiple parties to communicate with each other for the sake of a common goal without revealing more information than originally intended. As part of its ongoing efforts, the PEC project maintains an open draft for establishing a use case suite for privacy-enhancing cryptography [12].

Perhaps a broader notion with increasingly more traction in the recent both in academia and industry is the notion of privacy-enhancing technologies (PETs), which is certainly not new [13,14], however, it has been seen as a much needed approach to security and privacy.

PETs can be organized in two main categories:

- Techniques for computing over encrypted data (COED), which include:
  - Fully homomorphic encryption (FHE),
  - Multiparty computation (MPC), and
  - Verifiable Computation (VC)/Zero-Knowledge Proofs (ZKP),
- Techniques for privacy-preserving statistics, which include:
  - Differential privacy,
  - Synthetic data generation, and
  - Federated machine learning.

In this work we will focus on a technology that is useful for enabling one flavor of computation over encrypted data and for the sake of scope, we will briefly discuss the main goals of associated technologies while not addressing statistical techniques. More over, we will discuss threshold-secret sharing schemes which not only are useful for providing threshold security [15,16,17] but it can also enable multiparty computation.

1.1 | Our Contribution

In this work we propose a variation of a well-known threshold secret sharing scheme introduced by Shamir in 1979 [18]. More often than not, secret sharing schemes operate over fields such as $\mathbb{Z}_p$ (for $p \geq 2$ prime). These schemes can be certainly used for multidimensional objects but not with an overhead as a function of the realized dimensionality. Generalizations of secret sharing schemes for any access structure have been proposed [19,20,21,22,23] but not without a significant expansion of the original scheme's description, and/or a needed increase in the number of shares, and/or an added overhead in the computation of shares.

An interesting result [24] is proposed where secret sharing schemes and secure multiparty computation is enabled over small fields based on algebraic geometry. Although that proposal allows polynomials over vector spaces, the secret itself is an element of $\mathbb{F}_q$ and that is a key difference in comparison to our proposal.

We propose a scheme directly derived from Shamir's scheme in which we replace $\mathbb{Z}_p$ by a geometric product space of arbitrary dimension n which straightforwardly handles threshold secret sharing of multivectors and from well-known isomorphisms between matrix and geometric algebras (GAs) [25], our scheme can also work over matrices. Our construction does not increase complexity in its description nor requires addition number of shares in comparison with Shamir's scheme. Instead, we keep its description as untouched as possible while enabling secret sharing and secret reconstruction using multivectors.

We discuss the importance of threshold cryptography for security and threshold secret sharing schemes for enabling multiparty computation and how the GA community can take advantage of such tools to add well-known security properties to GA-based

applications with no changes in their algebraic structures. As we believe that multivectors can exceed its perceived utility to be a general-purpose data structure for arbitrary computations, we hope researchers inside and out the GA community can appreciate the unification of known-cryptographic mechanisms and GA in a straightforward fashion.

We show how our approach to threshold secret sharing enables, naturally, multiple-secret sharing in a single secret object as another direct benefit of working with multivectors, even for those not involved with GA.

We also show how linear and non-linear functions can be computed over secret shares and from these two operations one can compute any arithmetic circuit, which is the basis for general-purpose secure multiparty computation.

Moreover, to the best of our knowledge, this is the first proposal of threshold secret sharing with secret multivectors.

1.2 | Preliminaries

We begin by reviewing the key concepts that leads us to an informed view of threshold secret sharing, its importance and benefits, associated technologies, and the context in which they are applied.

Verifiable Computation/Zero-Knowledge

Assuming there is a way to securely compute on encrypted data, which implies that parties other than the data owner would be able to evaluate functions with ciphertext inputs without knowing what they were computing, how can the data owner(s) know if the outsourced computation was 1) indeed the expected function/algorithm and 2) that the computation itself was computed correctly? In PET, this can be done at least in two ways. If the party performing COED has no access to any secret information, then this can be done via VC. Otherwise, this can be done via ZKP. VC is discussed at length in[26,27,28,29], same as ZKP in[30,31,32,33]. Generally speaking, a VC scheme enables party A with limited computation to outsource to party B the evaluation of a function F on the input x. This is done in such a way that party A can verify the correctness of $F(x)$ with less computation than what is required to evaluate $F(x)$. Thus, VC only makes sense to party A is the computation of the verification is (much) less intensive than the computation of the associated verification. On ZKP, Goldreich[31] remarks that a proof that $x \in L$ is a "witness" w_x such that $P_L\left(x, w_x\right) = 1$ where P_L is a polynomial-time computable Boolean predicate associated to the language L $\left(\text{i.e., } P_L(x, y) = 0\ \forall y \text{ if } x \notin L\right)$. As an immediate requirement, the witness must have length polynomial in the length of the input x, without the need of being computable from x in polynomial-time. Furthermore, a proof system is said to be zero-knowledge if whatever the verifier could generate in probabilistic polynomial-time after "seeing" a proof of membership, he could also generate in probabilistic polynomial-time after simply being told by a trusted oracle that the input is indeed in the language.

Without going into the details of how VC and ZKP works, from this brief discussion we can clearly see what roles they play in PET, why they are needed, and when they make sense to be invoked. Overall, they exist to verify correctness of a given computation. But what computation? And how this computation takes place? To answer theses questions we need to talk about FHE and MPC.

Fully Homomorphic Encryption

FHE[6] is a technology that allows a data owner to encrypt their data and be the only one able to decrypt it at the same time that they are able to *delegate* computation over that encrypted data without the need of prior decryption. In theory, a FHE scheme allows correct addition and multiplication over encrypted data and from these two basic operations one can compute a circuit (Boolean or arithmetic) that evaluates *any function*. In practice, current schemes are able to efficiently compute functions that can be expressed as low multiplicative depth circuits or polynomials of low degree. Applications that mostly rely on linear functions and/or polynomials of low degree are candidates for using well-known FHE libraries[34,35,36,37,38].

Overall, an encryption is said to be a homomorphic encryption (HE) if for its encryption function f and for all $a, b \in \mathcal{P}$ where $\mathcal{P}$ is the plaintext space, it holds that $f(a \circ b) = f(a) \circ f(b)$ for some operation $\circ$. In other words, the encryption of an operation over unencrypted operands equals the operation of the individual encryptions of the operands. Generally speaking, a HE scheme is additive homomorphic if only allows addition, and mutiplicative if only allows multiplication over encrypted data. If a HE scheme allows both addition and multiplication, that scheme is said to be fully homomomorphic. More specifically, HE schemes are classified according to what type of circuits they allow, taking into consideration their gates, size, and depth. A detailed discussion on the various types of HE schemes, their properties and requirements is found in[39].

Multiparty Computation

MPC[40,41,42,43,44], also known as secure computation, achieves similar goals as the ones related to FHE. However, as the name implies, MPC requires multiple parties to evaluate a joint function. In MPC, security is typically establish through the fact that there are multiple parties computing over random shares of any given data input. Another similarity of FHE and MPC is that, generally speaking, computing linear functions is also a more involved process. While FHE requires large amounts of computation to compute linear functions, MPC requires more communication between the involved parties. MPC is generally more efficient than FHE even when computing complex functions. However, while FHE is supposed to handle arbitrary functions in arbitrary scenarios, MPC is typically designed to evaluate pre-defined functions of specific application scenarios.

Perhaps a good way to introduce MPC is to think about outputs that are computed by two parties via simple mechanisms. As an example, in 1989, den Boer proposed a two-party cryptographic protocol for evaluating any gate[45]. As an illustration of the capabilities of that protocol, the following scenario was discussed: Alice and Bob were trying to find out if they had any mutual interest while avoiding exposing their interest in the possibility of facing a non-reciprocal answer. They would however agree on engaging in such procedure if they knew that their answers would only be revealed if they both liked each other. This can be formalized as Alice having a secret bit a and Bob a secret bit b for which a protocol is needed in order to reveal exactly the bitwise AND of a and b. Alice and Bob would then set their bit to 0 if they did not like the other or 1 if otherwise. It is clear that the operation a AND b would only review 1 if both Alice and Bob set their secret bits to 1. In other words, if one party sets their answer to 0, they would not learn what the other party's answer was.

Although the bitwise AND seems to efficiently solve this problem, den Boer introduces another way of achieving it using five cards with the following configuration: the back of all cards must be the same, as usual. The face side of two cards are of both of the same type and the face of the three other cards are all of some other type. As an example, the configuration could be two two-of-hearts and three two-of-spades. The protocol would then include the following steps:

1. Each party receives one card of each type and the remaining space is left face down on the table.
2. Bob decides if he likes Alice. If the answer is yes, he will place the heart on the top; naturally, heart will go on bottom if otherwise. Then, Bob places his two cards on top of the initial spade on the table.
3. Alice decides if she likes Bob. If the answer is yes, she will put heart on the bottom; if otherwise, heart will go on the top. Then, Alice places her two cards on the bottom of the initial space on the table (that is, underneath all the other cards).
4. Bob and Alice alternate in cutting the cards as many times and in as many ways they want. Once they are done, they display the cards on the table in a circular fashion to ensemble a cyclic group.

Thinking in terms of cyclic permutations, there are only two possible distinct outcomes. We visualize this by considering the following scenarios. In the first scenario, Alice and Bob does not like each other.

These two examples are useful for illustrating the practical implications of using more than one party to compute a desired output while preserving the privacy with respect to the information the data owner wants to protect. What about more general computations? MPC schemes are typically enabled by secret sharing data, that is, given an input data, generate a number of shares that will match the number of parties involved in any given computation. Among many other requirements, it is expected that each party will only know their corresponding shares and the result of the functions they compute. A variant of secret sharing schemes is known as threshold secret sharing, which we discuss next.

1.3 | Organization

The remainder of this manuscript is organized as follows: In Section 2 we discuss the fundamentals of threshold secret sharing by examining Shamir's proposal. In Section 3 we introduce a variation of Shamir's scheme using multivectors. In Section 4 we present a concrete example of our construction. In Section 5 we discuss some of the immediate applications enable by our approach. In Section 6 we point to some of the next steps we envision as part of this ongoing investigation. In Section 7 we provide the conclusions.

2 | THRESHOLD SECRET SHARING

In 1979, Shamir introduced a scheme for dividing data D into n shares such that D is easily reconstructed from any t shares[18]. One particular feature of this scheme is that even total knowledge of $t-1$ shares reveals absolutely no information about D and therefore t is the *threshold* for reconstructing D. This solution is known as (t, n)-*threshold secret-sharing*.

As it happens with many other solutions in cryptography, there is a notion of a particular necessity that defines a more general problem that can be solved with a (t, n)-threshold secret-sharing scheme. As an example, the following problem is discussed in[46]:

> Eleven scientists are working on a secret project. They wish to lock up the documents in a cabinet so that the cabinet can be opened if and only if six or more of the scientists are present. What is the smallest number of locks needed? What is the smallest number of keys to the locks each scientist must carry?

In order to answer the two questions above, recall that $P(n,t) = \frac{n!}{(n-t)!}$ denotes t *permutations* of n distinct objects and $C(n,t) = \frac{P(n,t)}{t!} = \frac{n!}{t!(n-t)!}$ denotes t *combinations* of n distinct objects. The answer to the two questions above is provided by the notion of combination over the data we obtain from the problem. Although apparently simple, the solution require some discussion that we hope will serve to aid appreciation for a more elegant and efficient solution.

For answering the first question, consider the following: given two distinct group of five scientists, there exists at least one scientist that is able to unlock a locker $\mathcal{L}$, otherwise we can form a group of six scientists which cannot unlock $\mathcal{L}$, which is then a contradiction. For any 5-group, there is a unique $\mathcal{L}$ which they cannot open. The number of locks is then given by $C(11,5) = 462$, which is the answer to the first question. For answering the second question, fix a scientist s_0. The number of all sets S of five scientists excluding s_0 is given by $C(10,5) = 252$. Notice that for all S, S cannot open the cabinet while $S \cap \{s_0\}$ can, which means that s_0 must carry at least one key for each set S and therefore 252 is the answer to the second question.

Finding the right answers however is clearly not enough. Even with modest values for n and t, it is impractical to consider eleven scientists carrying 252 keys to open 462 lockers of a single cabinet.

Shamir's (t, n)-threshold secret-sharing scheme is a much more efficient solution to the problem described previously in which t is the number of required scientists (the threshold) to "unlock a secret" and n is the total number of scientists.

2.1 | Shamir's Secret Sharing (SSS)

Let F be a finite field with m elements. Recall the following (well-known) facts. Given $n \leq m$ distinct points in $F \times F$, there is a unique polynomial $q \in F[x]$ of degree $n+1$ which contains these points. Moreover, given only the $n+1$ points, the polynomial q can be constructed using polynomial interpolation. SSS relies heavily on these facts. We detail the scheme below:

Let p be a prime large enough to guarantee that the field $F = \mathbb{Z}_p$ contains all integer data needed for a particular application. Fix a secret $S \in F$ and generate a polynomial q of degree $t+1$ whose constant term is S $\left(\text{i.e. } q(0) = S\right)$ and whose remaining coefficients are randomly chosen from F. We generate $n \leq p$ shares of the secret by computing $q(i)$ for $i = 1, \ldots, n$. Each share of the secret is then a point $\left(i, q(i)\right) \in F \times F$. The polynomial q can be reconstructed (using Lagrange interpolation, for instance) provided at least t shares are known.

This scheme is secure in the sense that if a malicious party (seeking to find the secret S) has fewer than t shares, then for each $S^* \in F$ they can construct a polynomial (often many polynomials) q^* of degree $t+1$ whose constant term is S^*. It suffices to consider the "worst" case, in which the malicious party has exactly $t-1$ shares. In such case, the point $(0, S^*)$ along with the $t-1$ known shares induce a unique polynomial of degree $t+1$ whose constant term is S^*. This is the aforementioned q^*. Thus, the malicious party cannot discount any elements of F as possible values for the secret.

We can organize SSS as a tuple of three basic algorithms:

$$(\texttt{Setup}(\lambda, t, n), \texttt{ComputeShares}(s), \texttt{Reconstruct}(S))$$

such that

- `Setup` takes λ, t, and n as arguments and uniformly generate a random λ-bit prime p, and makes p, t and n accessible to `ComputeShares` and `Reconstruct`. By taking t, and n in `Setup` we assume that `ComputeShares` and `Reconstruct` will always work with the values t, and n defined at the time of the setup. A different configuration would require another setup.

- `ComputeShares` takes a secret $s \in \mathbb{Z}_p$ and proceeds as follows:
  1. Set $a_0 := s$
  2. Uniformly and independently generate $a_1, \ldots, a_{t-1} \leftarrow \mathbb{Z}_p$
  3. Construct the polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$
  4. Compute n shares s_i such that $s_i = (i, f(i) \bmod p)$, $i = 1, \ldots, n$
  5. Return $S = \left(s_1, \ldots, s_n\right)$.
- `Reconstruct` takes S as argument where S is a set of order t with any t out of the original n shares of s and proceeds as follows:
  1. Compute the Lagrange basis polynomial such that:
  $$\ell_j(x) = \prod_{\substack{0 \le i \le t \\ j \ne i}} \frac{x - x_i}{x_j - x_i} \tag{1}$$
  2. Organize the information in S as points such that
  $$\left(x_j, y_j\right) = \left(i, s_i\right) \tag{2}$$
  from $j = 0, \ldots, t-1$ and from the values of i in S.
  3. Compute a polynomial interpolation to reconstruct $f(x)$:
  $$f(x) = \sum_{j=0}^{t-1} y_j l_j(x) \tag{3}$$
  4. The secret is recovered by computing $f(0) = s$.
  5. Return s.

2.2 | Numerical Example

Let $\lambda = 9$, $t = 3$, and $n = 5$. So `Setup`(λ, t, n) generates $p = 367$ and make t and n accessible by `ComputeShares` and `Reconstruct`. Let the secret be $s = 150$ so `ComputeShares`(s) uniformly and independently generates $a_1 = 196$, and $a_2 = 144$ such that

$$f(x) = 150 + 196x + 144x^2. \tag{4}$$

We compute 5 shares s_i such that $s_i = (i, f(i) \bmod p)$, $i = 1, \ldots, 5$ such that

$$s_1 = (1, 123), s_2 = (2, 17), s_3 = (3, 199), s_4 = (4, 302), s_5 = (5, 326). \tag{5}$$

Now, say that we randomly select s_2, s_4, and s_3 to reconstruct s. We the let $S = \left(s_2, s_4, s_3\right)$ so `Reconstruct`(S) can compute (8), (9), and (10). Interesting to notice that we only need to compute $\ell_j(0)$ such that

$$\begin{aligned} \ell_0(0) &= \tfrac{0-4}{2-4} \cdot \tfrac{0-3}{2-3} = 6 \\ \ell_1(0) &= \tfrac{0-2}{4-2} \cdot \tfrac{0-3}{4-3} = 3 \\ \ell_2(0) &= \tfrac{0-2}{3-2} \cdot \tfrac{0-4}{3-4} = -8 \end{aligned} \tag{6}$$

We finally reconstruct the secret s by computing

$$s = y_0 \ell_0(0) + y_1 \ell_1(0) + y_2 \ell_2(0) = 150 \bmod 367. \tag{7}$$

3 | (T, Z)-THRESHOLD SECRET-SHARING WITH GA

We will now derive from SSS a threshold secret-sharing scheme using GA. For practicality and for better reconciling notations from secret-sharing schemes and GA, let's consider multivectors of Clifford signatures $\mathcal{C}\ell(n, 0)$ as elements of the geometric product space $\mathbb{G}^n$. Whenever all the computations on the coefficients of those multivectors are reduced modulo a prime p, we denote that geometric product space by $\mathbb{G}_p^n$. We will keep denoting the threshold by t and we now denote the number of total shares by z so we have a GA-based (t, z)-threshold secret-sharing.

We will use the same algorithms from SSS and redefine them for working with elements of $\mathbb{G}_p^n$ for some prime p.

- `Setup` takes λ, t, and z as arguments and uniformly generate a random λ-bit prime p, and makes p, t and n accessible to `ComputeShares` and `Reconstruct`.
- `ComputeShares` takes a secret $S \in \mathbb{G}_p^n$ and proceeds as follows:
  1. Uniformly and independently generate $A_1, \ldots, A_{t-1} \leftarrow \mathbb{G}_p^n$
  2. Construct the polynomial $f(x) = S + A_1 x + A_2 x^2 + \ldots + A_{t-1} x^{t-1}$
  3. Compute z shares s_i such that $s_i = f(i),\ i = 1, \ldots, z$
  4. Return $\left(s_1, \ldots, s_z\right)$.
- `Reconstruct` takes any t-length subset of $\left\{s_1, \ldots, s_z\right\}$ as argument and proceeds as follows:
  1. Compute the Lagrange basis polynomial such that:
  $$\ell_j(x) = \prod_{\substack{0 \le i \le t \\ j \ne i}} \frac{x - x_i}{x_j - x_i} \tag{8}$$
  2. Organize the information in S as points such that
  $$\left(x_i, y_i\right) = s_i,\ i = 0, \ldots, t-1. \tag{9}$$
  3. Compute a polynomial interpolation to reconstruct $f(x)$:
  $$f(x) = \sum_{j=0}^{t-1} y_j l_j(x) \tag{10}$$
  4. The secret is recovered by computing $f(0) = S \in \mathbb{G}_p^n$.
  5. Return S.

3.1 | Security

Our (t, z)-threshold secret sharing scheme is said to be perfect if knowledge of $t-1$ of fewer out of z shares provides no information about S and is said to be ideal if every share is of the same size (that is, comes from exactly the same space with precisely the same boundaries) as the secret S. Another interesting property is that a perfect and ideal threshold secret sharing scheme should not rely on security assumption, which are common in cryptography such as integer factorization, approximate greatest common divisor, discrete logarithm, among many others.

Shamir's schemes satisfies these three properties. It it easy to see that our scheme is ideal since shares are uniformly generated from $\mathbb{G}_p^n$, which is the space from all allowed values of S come from. But can we show that our scheme is also perfect?
We observe trivially that this scheme is ideal since the shares come from the same domain as the secret; namely $\mathbb{G}_p^n$. Further, the scheme is perfect for the same reason as Shamir's original scheme. We elaborate below:

Proposition 1. The above-defined (t, z)-threshold secret-sharing scheme is perfect in the sense that any collection of fewer than t shares yields no information about the secret.

Proof. We begin by showing that each t-tuple of shares corresponds to a unique polynomial of degree at most $t-1$. If $g = |\mathbb{G}_p^n|$, then there are g^t possible t-tuples of shares and g^t possible polynomials in $\mathbb{G}_p^n[x]$ of degree at most $t-1$. Uniqueness follows, else there will be a t-tuple of secrets which does not correspond to a polynomial of degree at most $t-1$.

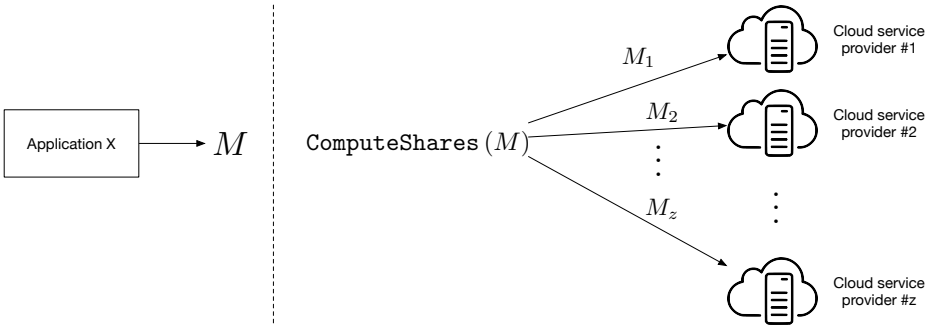

FIGURE 1 Threshold security for data storage.

Let $\sigma_1, \ldots, \sigma_{t-1}, \sigma$ be distinct shares with the σ_i fixed, and let $\ell_1, \ldots, \ell_t$ be the Lagrange basis polynomials. Note that these polynomials are fixed and each $\ell_i(0)$ is nonzero. Now, if S is the secret, then

$$S = \sigma \ell_t(0) + \sum_i \sigma_i \ell_i(0) \tag{11}$$

By the preceding uniqueness result and (11), there is a one-to-one correspondence between the possible values of the share σ and the possible values of the secret S. In particular, knowing only the shares $\sigma_1, \ldots, \sigma_{t-1}$ means an attacker has g polynomials in $\mathbb{G}_p^n[x]$ (one for each secret $S \in \mathbb{G}_p^n$) to choose from, and cannot eliminate any possible value for the secret. Furthermore, each value for the secret s is equally probable. □

The above results on idealness and perfectness show that in extending the domain of secrets for Shamir's scheme to $\mathbb{G}_p^n$, we maintain the same security and efficiency.

3.2 | Thinking Threshold Security

Threshold secret sharing schemes can have simple and yet solid implications for security. As an example, it is known that GA has an increasing number of applications in various engineering sciences[47,48,49,50,51,52]. In many of these applications, information is represented as multivectors, which could be geographic coordinates, image specifications, objects in animations and games, visual patterns, objects in robotics and machine learning, among many others. For all of these cases and more, after multivectors are created and/or computed on, they will eventually be stored in some non-volatile memory. This is precisely the moment where our scheme comes into play. Whatever piece of information in the form of multivectors can have random shares created for it where each share is sent to a distinct destination.

Figure 1 illustrates and simple and yet powerful use case of threshold security with multivectors. Let X be an application that handles pieces of information in multivector form. When the data generation and/or data manipulation represented as, say, M, is complete and M is intended to be retrieved at a later time, we can use our threshold secret sharing scheme to compute z shares of M and send each share to a distinct location. These locations can be distinct cloud service providers (CSPs), which are the parties involved. We think of a single machine in each CSP. If one provisions more machines in a single CSP and that CSP is compromised (either due to malicious activities or to malfunction), then all machines in that CSP are automatically compromised.

We then have z shares, for $z > t$, spread across multiple CSPs, while we only need t shares for reconstructing M. To achieve desired threshold security, the concrete values of t and z are defined according to the likelihood of t parties being compromised at the same time. The compromise of up to $t - 1$ parties does not reveal anything about M.

To highlight how threshold security is interesting in many ways, we go back to our initial discussion about the CIA triad. Very feel single cryptographic solutions can address more than one of the three principles at once. For proper values of t and z, which directly depend on each type of application and a previously assessed likelihood of compromise of $t - 1$ parties, shares of M are stored in separate CSPs where each share is in possession of random data and each party is assumed to only know its share, we have confidentiality. There are more shares available than the required number for reconstructing M. This property allows us to lose some of the parties and as long as we have t parties holding their shares, we can reconstruct M. Then we also have availability.

3.3 | Thinking Multiparty Computation

Threshold secret sharing scheme as Shamir's are known as linear secret sharing schemes (LSSS)[53], that is, a secret sharing scheme with reconstruction f the secret from shares as a linear mapping. A direct implication of LSSS is that any linear computation evaluated on the individual shares by each of their associated parties (also known as local computation, since each party is locally evaluating functions on their own shares), will correspond to operations evaluated directly on the secret. This is a form of additive homomorphism. No extra communication is required between the parties. This property enables secure multiparty computation based on linear functions.

3.3.1 | Addition

As mentioned previously, addition on shares generated with any LSSS can be locally computed and the reconstruction of the result shares will correspond to the same operation of the original inputs. More specifically, let A and B be secrets and $\left(A_1, \ldots, A_z\right)$ and $\left(B_1, \ldots, B_z\right)$ be the secret shares of A and B, respectively. Given z parties, each party is given the pair $\left(A_i, B_i\right)$. They compute $C_i = A_i + B_i \in \mathbb{G}_p^n$ and via reconstruction, we obtain

$$f(0) = \sum_{j=0}^{t-1} y_j l_j(0) \equiv A + B \in \mathbb{G}_p^n. \tag{12}$$

3.3.2 | Multiplication

For secrets A and B with respective secret shares, $\left(A_1, \ldots, A_z\right)$ and $\left(B_1, \ldots, B_z\right)$, it is possible to calculate the product AB. Since multiplying two polynomials of equal degree results in a polynomial of twice that degree, if the threshold for recovering either A or B is t then the threshold for calculating the product AB will be $2t$. The shares for AB can be locally computed, $C_i = A_i B_i$ and the product reconstructed by

$$f(0) = \sum_{j=0}^{2t-1} C_j l_j(0) \equiv AB \in \mathbb{G}_p^n \tag{13}$$

Furthermore it is possible to multiply the result, AB, a second time by recomputing the shares of AB locally though this requires each party to know the number of shares and the threshold for the scheme.

4 | A CONCRETE EXAMPLE

We now fix multivectors in $C\ell(3,0)$, which are members of the 3-dimensional geometric product space where computation on multivectors coefficients are reduced modulo p. We denote this space by $\mathbb{G}_p^3$ for p prime. For all multivectors $M \in \mathbb{G}_p^3$, we write

$$M = m_0\mathbf{e}_0 + m_1\mathbf{e}_1 + m_2\mathbf{e}_2 + m_3\mathbf{e}_3 + m_{12}\mathbf{e}_{12} + m_{13}\mathbf{e}_{13} + m_{23}\mathbf{e}_{23} + m_{123}\mathbf{e}_{123}. \tag{14}$$

Setup

In terms of implementation, `Setup` (λ, t, z) defines $\lambda = 9$, $t = 3$, $n = 5$ and makes them available to `ComputeShares` and `Reconstruct`.

Computing shares

Now we let the secret S be

$$S = 176\mathbf{e}_0 + 173\mathbf{e}_1 + 196\mathbf{e}_2 + 114\mathbf{e}_3 + 54\mathbf{e}_{12} + 73\mathbf{e}_{13} + 16\mathbf{e}_{23} + 7\mathbf{e}_{123}. \tag{15}$$

We then invoke `ComputeShares` (S) which generates A_1 and A_2 such that

$$\begin{aligned} A_1 &= 100\mathbf{e}_0 + 29\mathbf{e}_1 + 173\mathbf{e}_2 + 28\mathbf{e}_3 + 159\mathbf{e}_{12} + 254\mathbf{e}_{13} + 99\mathbf{e}_{23} + 214\mathbf{e}_{123}, \\ A_2 &= 236\mathbf{e}_0 + 239\mathbf{e}_1 + 95\mathbf{e}_2 + 29\mathbf{e}_3 + 150\mathbf{e}_{12} + 119\mathbf{e}_{13} + 245\mathbf{e}_{23} + 142\mathbf{e}_{123}. \end{aligned} \tag{16}$$

and proceeds to creates the following shares of S:

$$\begin{aligned}
s_1 &= 1,\ f(1) = S_1 = 255\mathbf{e}_0 + 184\mathbf{e}_1 + 207\mathbf{e}_2 + 171\mathbf{e}_3 + 106\mathbf{e}_{12} + 189\mathbf{e}_{13} + 103\mathbf{e}_{23} + 106\mathbf{e}_{123},\\
s_2 &= 2,\ f(2) = S_2 = 35\mathbf{e}_0 + 159\mathbf{e}_1 + 151\mathbf{e}_2 + 29\mathbf{e}_3 + 201\mathbf{e}_{12} + 29\mathbf{e}_{13} + 166\mathbf{e}_{23} + 232\mathbf{e}_{123},\\
s_3 &= 3,\ f(3) = S_3 = 30\mathbf{e}_0 + 98\mathbf{e}_1 + 28\mathbf{e}_2 + 202\mathbf{e}_3 + 82\mathbf{e}_{12} + 1079\mathbf{e}_{13} + 205\mathbf{e}_{23} + 128\mathbf{e}_{123},\\
s_4 &= 4,\ f(4) = S_4 = 240\mathbf{e}_0 + 1\mathbf{e}_1 + 95\mathbf{e}_2 + 176\mathbf{e}_3 + 6\mathbf{e}_{12} + 166\mathbf{e}_{13} + 220\mathbf{e}_{23} + 51\mathbf{e}_{123},\\
s_5 &= 5,\ f(5) = S_5 = 151\mathbf{e}_0 + 125\mathbf{e}_1 + 95\mathbf{e}_2 + 208\mathbf{e}_3 + 230\mathbf{e}_{12} + 206\mathbf{e}_{13} + 211\mathbf{e}_{23} + 1\mathbf{e}_{123},
\end{aligned} \tag{17}$$

where

$$s_1 = (1, S_1), s_2 = (2, S_2), s_3 = (3, S_3), s_4 = (4, S_4), s_5 = (5, S_5). \tag{18}$$

Reconstructing the secret

In order to reconstruct S, we need a list from $\{s_1, \ldots, s_z\}$ with length t in any order. Let this list be

$$(s_4, s_2, s_3) = ((4, S_4), (2, S_2), (3, S_3)). \tag{19}$$

For the sake of matching notation of the reference scheme and maket easier to read, let

$$\begin{aligned}
(x_0, x_1, x_2) &= (4, 2, 3),\\
(y_0, y_1, y_2) &= (S_4, S_2, S_3).
\end{aligned} \tag{20}$$

Now, we compute $\ell_j()$ such that

$$\begin{aligned}
\ell_0(0) &= \tfrac{0-x_1}{x_0-x_1} \cdot \tfrac{0-x_2}{x_0-x_2} = \tfrac{0-2}{4-2} \cdot \tfrac{0-3}{4-3} = 3,\\
\ell_1(0) &= \tfrac{0-x_0}{x_1-x_0} \cdot \tfrac{0-x_2}{x_1-x_2} = \tfrac{0-4}{2-4} \cdot \tfrac{0-3}{2-3} = 6,\\
\ell_2(0) &= \tfrac{0-x_0}{x_2-x_0} \cdot \tfrac{0-x_1}{x_2-x_1} = \tfrac{0-4}{3-4} \cdot \tfrac{0-2}{3-2} = -8,
\end{aligned} \tag{21}$$

which completes everything we need to know in order to reconstruct S, which is given by evaluating $f(0)$:

$$f(0) = y_0\ell_0(0) + y_1\ell_1(0) + y_2\ell_2(0) = S_4 3 + S_2 6 + S_3(-8) \tag{22}$$

which gives

$$f(0) = S = 176\mathbf{e}_0 + 173\mathbf{e}_1 + 196\mathbf{e}_2 + 114\mathbf{e}_3 + 54\mathbf{e}_{12} + 73\mathbf{e}_{13} + 16\mathbf{e}_{23} + 7\mathbf{e}_{123}, \tag{23}$$

so we correctly reconstructed S from t out of z shares.

5 | KEY DIFFERENTIATORS

In proposing and analyzing our approach, we were motivated by expanding Shamir's original scheme with the least disruptive modifications so that we could preserve the properties that makes Shamir's contribution so interesting. At the same time we wanted to make sure this subtle modification would still expand the original set features so that one could achieve more without inducing too much overhead. In this section we discuss some of the benefits of working with our construction.

5.1 | Immediate Benefits

Shamir's secret sharing scheme rely on the coefficients of a polynomial forming an algebra over an appropriate field. We can then summarize our approach to threshold secret sharing as using GA as a special case of Shamir's scheme since $\mathbb{G}_p^n$ qualifies to be such algebra. This is particularly interesting since changing any other component of Shamir's original scheme could risk some of its fundamental properties such as perfect secrecy and perfect idealness, which would immediately require a new stream of analysis to support claims of property equivalence. One of the motivation of our experiments can be expressed in the following question: how can we expand the functionality of Shamir's scheme with the least intrusive modification? Hence, our proposal.

As we mentioned before, on top of all well-known specialized applications of GA, we believe that GA can be explored as a general-purpose algebraic structure and consequently, multivectors can be general-purpose data structures. Therefore our approach to threshold secret sharing can be unintrusively be used by anyone dealing with information in the geometric product space, vector space, and even matrix space (via isomorphisms between matrix and geometric algebras).

However, even those not involved with GA at all can still be directly benefited by our approach since we can use it as an alternative to *multi-secret sharing* or *packed secret sharing*, which we discuss next.

5.2 | An Alternative Method for Multi-Secret Sharing

The notion of multi-secret sharing or packed secret sharing is discussed by Franklin[54]. In short, for proper polynomails of degree up to $d-1$ where the total of corrupt parties is at most t, then one can use single polynomial to compute shares with respect to at most $d-t$ secrets. Using this strategy, if one wants to share k secrets, then k degrees of freedom are required. Notice that although d coefficients implies d degrees of freedom, t shares are enough to reconstruct at least one secret. So therefore the $d-t$ bound for the maximum allowed number of secrets.

With our approach, given any proper $\mathbb{G}_p^n$, that is, $n \geq 1$ and $p \geq 2$ prime, we can leave the fundamental polynomial structure untouched while allowing creating shares of as many secrets as desired. Multivectors in $\mathbb{G}_p^n$ have 2^n coefficients and therefore those working over the integers are allowed to have 2^n secrets. As an example, if we work with $\mathbb{G}_p^3$, then we are allowed to have up to 8 secrets. Let $s, t, u, v, w, x, y, z \in \mathbb{Z}_p$ be secrets. We have

$$S = s\mathbf{e}_0 + t\mathbf{e}_1 + u\mathbf{e}_2 + v\mathbf{e}_3 + w\mathbf{e}_{12} + x\mathbf{e}_{13} + y\mathbf{e}_{23} + z\mathbf{e}_{123} \tag{24}$$

and therefore we generate z shares by evaluating

$$f(x) = S + A_1 x + A_2 x^2 + \ldots + A_{t-1} x^{t-1} \tag{25}$$

for $x = 1, \ldots, z$.

This change is what one can label as a representational change, that is, the change is being made on the meaning of the secret without changing how the secret is fundamentally handled with respect to the desired properties of Shamir's original scheme.

For generalizing this notion, we have: Given $\mathbb{G}_p^n$, we can compute shares for up to 2^n secrets in $\mathbb{Z}_p$.

6 | FUTURE DIRECTIONS

This work is a non-exhaustive analysis of threshold secret sharing with GA. As mentioned earlier, we use Shamir's scheme as a reference for yielding a variation through the least of the possible modifications we encountered. It is known that computing on shares reveals some level of information. For instance, at the very least, the parties learn the output of their local computations and the output of other parties when computing a joint function requires communication between parties. In order to use our construction for multiparty computation, further analysis of what level information is learned by the parties must be further examined, specially before any serious implementation in the real world. An in-depth comparison between our approach and other approaches somehow involving vector and matrices can also review trade-offs with respect to computational and space efficiency.

7 | CONCLUSIONS

In this work we introduce a variant of the well-known threshold secret sharing scheme proposed by Adi Shamir in 1979. Our construction enables multidimensional objects such as multivectors to be secrets in a secret sharing scenario without incurring in any additional overhead in comparison with our reference scheme. Our construction allows geometric Algebra (GA) practitioners and any other audience working with multivectors to explore the benefits of threshold security. We show that fundamental computations such as addition and multiplication can be performed over random shares and therefore, as one of the natural next steps, our scheme can be extended to a fully functional multiparty computation protocol with GA. We show how our proposal can also serve as an alternative solution for multi-secret sharing.

References

1. Rivest Ronald L, Adleman Len, Dertouzos Michael L, others . On data banks and privacy homomorphisms. *Foundations of secure computation.* 1978;4(11):169–180.
2. ElGamal Taher. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory.* 1985;31(4):469–472.
3. Paillier Pascal. Public-key cryptosystems based on composite degree residuosity classes. In: :223–238Springer; 1999.
4. Goldwasser Shafi, Micali Silvio. Probabilistic encryption. *Journal of computer and system sciences.* 1984;28(2):270–299.
5. Cohen Josh D, Fischer Michael J. *A robust and verifiable cryptographically secure election scheme*. Yale University. Department of Computer Science; 1985.
6. Gentry Craig. Fully homomorphic encryption using ideal lattices. In: :169–178; 2009.
7. Gentry Craig, Halevi Shai. Implementing gentry's fully-homomorphic encryption scheme. In: :129–148Springer; 2011.
8. Van Dijk Marten, Gentry Craig, Halevi Shai, Vaikuntanathan Vinod. Fully homomorphic encryption over the integers. In: :24–43Springer; 2010.
9. Coron Jean-Sébastien, Naccache David, Tibouchi Mehdi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In: :446–464Springer; 2012.
10. Brakerski Zvika, Vaikuntanathan Vinod. Efficient fully homomorphic encryption from (standard) LWE. *SIAM Journal on Computing.* 2014;43(2):831–871.
11. Cheon Jung Hee, Kim Andrey, Kim Miran, Song Yongsoo. Homomorphic encryption for arithmetic of approximate numbers. In: :409–437Springer; 2017.
12. NIST CTG. Privacy-Enhancing Cryptography | CSRC https://csrc.nist.gov/projects/pec(Accessed on 01/29/2022); .
13. Goldberg Ian, Wagner David, Brewer Eric. Privacy-enhancing technologies for the internet. In: :103–109IEEE; 1997.
14. Burkert Herbert, others . Privacy-enhancing technologies: Typology, critique, vision. *Technology and privacy: The new landscape.* 1997;:125–142.
15. Desmedt Yvo G. Threshold cryptography. *European Transactions on Telecommunications.* 1994;5(4):449–458.
16. Kaya Kamer, Selçuk Ali Aydın. Threshold cryptography based on Asmuth–Bloom secret sharing. *Information sciences.* 2007;177(19):4148–4160.
17. Gennaro Rosario, Rabin Michael O, Rabin Tal. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: :101–111; 1998.
18. Shamir Adi. How to share a secret. *Communications of the ACM.* 1979;22(11):612–613.
19. Ito Mitsuru, Saito Akira, Nishizeki Takao. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science).* 1989;72(9):56–64.
20. Yang Chou-Chen, Chang Ting-Yi, Hwang Min-Shiang. A (t, n) multi-secret sharing scheme. *Applied Mathematics and Computation.* 2004;151(2):483–490.
21. Pang Liao-Jun, Wang Yu-Min. A new (t, n) multi-secret sharing scheme based on Shamir's secret sharing. *Applied Mathematics and Computation.* 2005;167(2):840–848.
22. Shao Jun, Cao Zhenfu. A new efficient (t, n) verifiable multi-secret sharing (VMSS) based on YCH scheme. *Applied Mathematics and Computation.* 2005;168(1):135–140.

23. Zhao Jianjie, Zhang Jianzhong, Zhao Rong. A practical verifiable multi-secret sharing scheme. *Computer Standards & Interfaces.* 2007;29(1):138–141.

24. Chen Hao, Cramer Ronald. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In: :521–536Springer; 2006.

25. Lounesto Pertti. *Clifford algebras and spinors*. Cambridge university press; 2001.

26. Parno Bryan, Howell Jon, Gentry Craig, Raykova Mariana. Pinocchio: Nearly practical verifiable computation. In: :238–252IEEE; 2013.

27. Costello Craig, Fournet Cédric, Howell Jon, et al. Geppetto: Versatile verifiable computation. In: :253–270IEEE; 2015.

28. Parno Bryan, Raykova Mariana, Vaikuntanathan Vinod. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In: :422–439Springer; 2012.

29. Chen Xiaofeng, Li Jin, Weng Jian, Ma Jianfeng, Lou Wenjing. Verifiable computation over large database with incremental updates. *IEEE transactions on Computers.* 2015;65(10):3184–3195.

30. Rackoff Charles, Simon Daniel R. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In: :433–444Springer; 1991.

31. Goldreich Oded, Micali Silvio, Wigderson Avi. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM (JACM).* 1991;38(3):690–728.

32. Goldreich Oded, Oren Yair. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology.* 1994;7(1):1–32.

33. Goldreich Oded, Krawczyk Hugo. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing.* 1996;25(1):169–192.

34. Microsoft SEAL (release 3.7) https://github.com/Microsoft/SEALMicrosoft Research, Redmond, WA.; 2021.

35. PALISADE . PALISADE Homomorphic Encryption Software Library – An Open-Source Lattice Crypto Software Library https://palisade-crypto.org/(Accessed on 01/29/2022); .

36. HElib . shaih/HElib: An Implementation of homomorphic encryption https://github.com/shaih/HElib(Accessed on 01/29/2022); .

37. lducas/FHEW https://github.com/lducas/FHEW(Accessed on 01/29/2022); .

38. Chillotti Ilaria, Gama Nicolas, Georgieva Mariya, Izabachène Malika. *TFHE: Fast Fully Homomorphic Encryption Library.* https://tfhe.github.io/tfhe/; August 2016.

39. Gentry Craig. Computing arbitrary functions of encrypted data. *Communications of the ACM.* 2010;53(3):97–105.

40. Shamir Adi, Rivest Ronald L, Adleman Leonard M. Mental poker. In: Springer 1981 (pp. 37–43).

41. Yao Andrew C. Protocols for secure computations. In: :160–164IEEE; 1982.

42. Goldreich Oded, Micali Silvio, Wigderson Avi. How to play any mental game, or a completeness theorem for protocols with honest majority. In: 2019 (pp. 307–328).

43. Chaum David, Damgård Ivan B, Graaf Jeroen. Multiparty computations ensuring privacy of each party's input and correctness of the result. In: :87–119Springer; 1987.

44. Chaum David, Crépeau Claude, Damgard Ivan. Multiparty unconditionally secure protocols. In: :11–19; 1988.

45. Boer Bert. More efficient match-making and satisfiability the five card trick. In: :208–217Springer; 1989.

46. White Lee J. Introduction to combinatorial mathematics (cl liu). *SIAM Review.* 1969;11(4):634.

47. Perwass Christian, Edelsbrunner Herbert, Kobbelt Leif, Polthier Konrad. *Geometric algebra with applications in engineering*. Springer; 2009.

48. Dorst Leo, Doran Chris, Lasenby Joan. *Applications of geometric algebra in computer science and engineering*. Springer Science & Business Media; 2012.

49. Doran Chris, Gullans Steven R, Lasenby Anthony, Lasenby Joan, Fitzgerald William. *Geometric algebra for physicists*. Cambridge University Press; 2003.

50. Corrochano Eduardo Bayro, Sobczyk Garret. *Geometric algebra with applications in science and engineering*. Springer Science & Business Media; 2001.

51. Dorst Leo, Mann Stephen. Geometric algebra: a computational framework for geometrical applications. *IEEE Computer Graphics and Applications.* 2002;22(3):24–31.

52. Hildenbrand Dietmar. Foundations of geometric algebra computing. In: :27–30American Institute of Physics; 2012.

53. Beimel Amos. Secret-sharing schemes: A survey. In: :11–46Springer; 2011.

54. Franklin Matthew, Yung Moti. Communication complexity of secure computation. In: :699–710; 1992.

AUTHOR BIOGRAPHY

David W. H. A. da Silva is a Senior Research Scientist at Symetrix Corporation and Algemetric and responsible for research and development of Algemetric products. David research and develops solutions related to security and privacy and efficient computation through applied mathematics. David started his career as a Software Engineer focused on web services and agile software development, which took him to be involved with several RD projects from start ups to goverment and large corporations. David has a Bachelor's degree in Business Adminsitration from Universidade Potiguar, Brazil, a Master's and a Ph.D degree in Computer Science from the University of Colorado at Colorado Springs, USA.

Luke Harmon received his Ph.D. in Applied Mathematics from the University of Colorado Colorado Springs in 2020 and has been an invited speaker at multiple universities and the American Mathematical Society meetings. Luke is responsible for the mathematical analysis, formal descriptions, and correctness proofs for protocols proposed by Algemetric. This includes finding flaws or vulnerabilities in the proposed mathematical designs and the study of proposed corrections.

Gaetan Delavignette received his M.S. in Applied Mathematics from the University of Colorado Colorado Springs 2013 and has served there as a professor. Gaetan is responsible for mathematical analysis, simulations, and formal descriptions for protocols proposed by Algemetric.