An Overview of Cryptographic Accumulators

Ilker Ozcelik[1,3][a], Sai Medury[1,2][b], Justin Broaddus[1,2][c] and Anthony Skjellum[1,2][d]

[1]*SimCenter, University of Tennessee at Chattanooga, Chattanooga, U.S.A.*

[2]*Department of Computer Science and Engineering, University of Tennessee at Chattanooga, Chattanooga, Tennessee, U.S.A.*

[3]*Department of Computer Engineering, Recep Tayyip Erdogan University, Turkey*

Keywords: Cryptographic Accumulator, Membership Test, RSA, Merkle Tree.

Abstract: This paper contributes a primer on cryptographic accumulators and how to apply them practically. A cryptographic accumulator is a space- and time-efficient data structure used for set membership tests. Since it is possible to represent any computational problem where the answer is yes or no as a set-membership problem, cryptographic accumulators are invaluable data structures in computer science and engineering. But, to the best of our knowledge, there is neither a concise survey comparing and contrasting various types of accumulators nor a guide for how to apply the most appropriate one for a given application. Therefore, we address that gap by describing cryptographic accumulators while presenting their fundamental and so-called optional properties. We discuss the effects of each property on the given accumulator's performance in terms of space and time complexity, as well as communication overhead.

1 INTRODUCTION

There are many use cases where one might need to maintain a list of elements with the purpose of determining whether an element being presented is part of this list or not. A common example is a list of credentials that have been authorized and granted certain privileges like an Access-Control List (ACL). During Authentication, an account management system will check to see whether the credentials entered are a part of the ACL or not and grant/deny privileges accordingly. If this list were small with only a few hundreds of elements at any given time, it would not take long to load the entire list into memory, compare each credential, and search for a match. The time complexity of this algorithm scales linearly ($O(n)$) with the size (n) of the list; therefore, it will perform poorly if the list grows to a number in the hundreds of thousands (and the performance is controlled by I/O speed, not main memory speed if the list is sufficiently large). We can reduce this complexity to sublinear ($O(\log n)$) by doing certain pre-computations on the list, such as by ordering it, then performing a binary search. Sublinear time complexity is an acceptable computational complexity by Industry standards but there is substantial overhead of sorting the elements that has the average computational complexity of $O(n \log n)$, which increases the total computational complexity of the algorithm to $O(n \log n)$.

We can further reduce computation complexity by trading off memory space by constructing auxiliary data structures like hashmaps with constant time lookup complexity. This could be a great alternative to pre-computation overhead and facilitates constant time lookup, which is literally the best possible speed up. However, this approach comes with an overhead of having to store extra data in memory that will also scale linearly ($O(n)$) with the size of the list. Depending on the memory size of the processing unit, a SHA256 hashmap representing a list that contains 10 million elements may not fit in the memory of a low-energy and resource constrained devices.

A cryptographic accumulator can be used as an alternative to search-based approaches. Cryptographic accumulators are space-efficient data structures that rely on cryptographic primitives to achieve sublinear time complexity for set membership operations. They were first proposed to solve document time-stamping and membership testing purposes (Benaloh and De Mare, 1993) Later, they were employed to implement authenticated data structures (Ghosh et al.,

[a] https://orcid.org/0000-0002-2032-1640
[b] https://orcid.org/0000-0002-2349-8783
[c] https://orcid.org/0000-0001-8540-997X
[d] https://orcid.org/0000-0001-5252-6600

Ozcelik, I., Medury, S., Broaddus, J. and Skjellum, A.
An Overview of Cryptographic Accumulators.
DOI: 10.5220/0010337806610669
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 661-669
ISBN: 978-989-758-491-6

2014; Goodrich et al., 2002) privacy-preserving (Slamanig, 2012) and anonymity-conscious (Camenisch and Lysyanskaya, 2002; Miers et al., 2013; Sudarsono et al., 2011) applications. Also, with the advent of blockchain technology, demand for data to be stored in a decentralized manner is rising rapidly and cryptographic accumulators are well suited to support quick constant-time set-membership testing on such data.

Cryptographic accumulators use novel probabilistic data structures for set-membership that minimize space complexity by compressing the hashmap using a set of cryptographic functions into a constant size bit-based data structures like the Bloom filter (Bloom, 1970), or its more recently proposed alternative called the cuckoo filter (Fan et al., 2014). This compression is lossy, inevitably giving rise to false positives from possible collisions.

Alternatively, one can use a cryptographic accumulator, a hash representative generated from the elements in the list, that is of constant size and provides constant time lookup complexity without potential false positives (Benaloh and De Mare, 1993). These types of cryptographic accumulators are better suited for distributed applications where a trusted authority is responsible for continuously maintaining the accumulator (hash representative); the trusted authority keeps the accumulator up to date with additions and deletions of elements from the list. Participants other than this authority could be clients themselves that are part of the list or else be verifiers that are trying to determine whether an element being presented is part of the list. Such cryptographic accumulators, also called asymmetric cryptographic accumulators (Kumar et al., 2014), additionally require the generation of witnesses (Kumar et al., 2014) for each element.

The witness is the value corresponding to an element that is required to verify an element's membership in the accumulator (Kumar et al., 2014); it is unique to each element and needs to be updated each time an element is added or deleted to the accumulator. This witness can also be communicated to the client for storage and later presentation on-demand to a verifier. The trusted authority, often known as the Accumulator Manager (AM) (Akkaya and Cebe, 2018), needs to perform $O(n)$ operations to update the witnesses of n elements for each new addition (resp, deletion) of an element to (resp, from) the list; this is the tradeoff for having virtually constant size computation complexity and communication overhead between the verifier, client, and AM for determining set-membership. Most studies thus far regarding cryptographic accumulators have been theoretical and focus on their underlying theories. By carefully combing through these studies and presenting a guide with a gentler learning curve, this paper provides an accessible discussion for those who are interested in learning, developing, and utilizing cryptographic accumulators in their applications.

The remainder of this paper is organized as follows: Section 2 introduces the set membership problem and cryptographic accumulators. Different architectures of cryptographic accumulators and their classification are also presented in this section. Effects of these optional features on accumulator operations are discussed in Section 3, in terms of memory usage, computational- and communicational-complexity. Finally, Section 4 concludes with a discussion of current and potential applications.

2 CRYPTO-ACCUMULATORS

In computer science and engineering, set membership tests (yes/no) tests) are used in many applications including database, authentication, and validation systems.

Testing set membership can be performed by running a search on the set but this method can be a resource intensive task as the set size increases. To address these limitations, researchers proposed cryptographic accumulators (Fan et al., 2014; Baldimtsi et al., 2017; Tremel, 2013; Reyzin and Yakoubov, 2016). The fundamental idea behind the accumulator is being able to accumulate values of a set A into a small value z in such a way that it is possible to prove only the elements of set A have been accumulated (Fazio and Nicolosi, 2002).

2.1 Classification

Cryptograph accumulators can be categorized as symmetric or asymmetric. Symmetric Accumulators (Kumar et al., 2014) are designed using symmetric cryptographic primitives and can verify membership of elements without the need of a corresponding witness. The Bloom filter (Bloom, 1970)—a type of array data structure—is a symmetric cryptographic accumulator that uses k-number of hash functions that set a unique combination of indices in the array based on the input element; it provides a limited representation of set-membership with a false positive rate that grows as the number of elements in the list approaches the max capacity of the list (Bloom, 1970). Equation 1 provides the estimate of the false positive rate of a simple Bloom filter construction:

$$FPR = (1 - [1 - \frac{1}{m}]^{kn})^k \approx (1 - e^{\frac{-kn}{m}})^k \qquad (1)$$

with m being the size of the array, k being the number of hash functions, and n being the number of accumulated elements. Variations of the Bloom filter (Fan et al., 2014) have sought to minimize this false positive rate but are unable to eliminate it entirely. Because Bloom filters are static accumulators, they cannot accommodate growing list sizes, so they must be regenerated after reaching full capacity during the transaction discovery process

The recently proposed cuckoo filter is a dynamic data structure that functions similarly to the simple Bloom filter but with additional capabilities such as the ability to delete elements. A cuckoo filter implements a cuckoo hash table (Fan et al., 2014) to save fingerprint representations of the elements in an accumulated set. A cuckoo hash table is an array of buckets in which each stored element is associated with two indices in the array. Two associated indices allow for the dynamic rearrangement of the elements stored in the cuckoo hash table, providing optimized space efficiency and low false-positive rates. For a given number of elements, a cuckoo filter outperforms a space-optimized Bloom filter in terms of false-positive rate and space overhead (Fan et al., 2014).

Asymmetric Accumulators (Kumar et al., 2014) require witness creation and update for dynamic verification of set membership (Baldimtsi et al., 2017). They are built on asymmetric cryptographic primitives (Baldimtsi et al., 2017) and require the underlying hash algorithm to exhibit the quasi-commutative property (Benaloh and De Mare, 1993). The RSA accumulator (Benaloh and De Mare, 1993) is one example of asymmetric accumulator that uses RSA modular exponentiation to achieve the quasi-commutative property. A simple RSA accumulator construction consists of the following expression for addition: $acc_n = acc_{n-1}^x \mod N$ where acc_n is the new accumulator value after addition, acc_{n-1} is the old accumulator value before addition, x is the element being added, and $N = pq$ where p and q are considered to be strong prime numbers whose Sophie Germain prime numbers $p' = \frac{p-1}{2}$ and $q' = \frac{q-1}{2}$ are the accumulator trapdoor. One drawback of an RSA accumulator is that it is collision-free only when accumulating prime numbers. A prime representative generator is required to accumulate composite numbers without collision . The implementation of an RSA accumulator by (Tremel, 2013) a random oracle prime representative generator provided by (Barić and Pfitzmann, 1997) was used. Asymmetric accumulators can be further classified based on operations supported and the type of membership proofs provided. This classification will be further explained and analyzed in the following sections.

A Merkle hash tree can also be implemented as an asymmetric accumulator to prove Set Membership of elements (Baldimtsi et al., 2017). It is classified as an asymmetric accumulator because a member of its set requires a witness to prove membership (or non-membership). But, it does not use asymmetric cryptographic primitives nor does it need the underlying hash function to exhibit quasi-commutative property. The root node of the Merkle Tree is called the Merkle root and its value is the pairwise accumulated hash of all of the non-root nodes in the tree. The Merkle root value must be recalculated when there is an addition/deletion of a member in the set. Checking for set membership can be done with a portion of the tree (Merkle, 1989), making it unnecessary to download the full data structure.

2.2 Architectures

There are two well known cryptographic accumulator architectures: the one-way and the collision-free accumulator.

2.2.1 One-way Accumulator

(Benaloh and De Mare, 1993) proposed the first-ever accumulator construction, known as the *One-way Accumulator*, characterized as a family of one-way hash functions with the additional quasi-commutative property. A one-way hash function H is a function that can accept an arbitrarily large message M and returns a constant size output that is also called a message digest MD. For H to be a one-way hash, it must satisfy the following properties (Weisstein, a; Weisstein, b):

- The description of function H should be public and should work without needing to know secret information.
- For a message M, it must be easy to calculate its message digest, $MD = H(M)$.
- Given an MD, it must be difficult to determine M for a range where H is valid.
- For any M, the probability of finding an $M' \neq M$ such that $H'(M) = H(M)$ is negligible.

The quasi-commutative property is a generalization of the commutative property. A function h defined as $h : X \star Y \Rightarrow X$ holds the quasi-commutative (Benaloh and De Mare, 1993) if $\forall x \in X$ and $\forall y_1, y_2 \in Y$,

$$h(h(x, y_1), y_2) = h(h(x, y_2), y_1). \qquad (2)$$

Despite using a strong one-way hash function (Fazio and Nicolosi, 2002), Benaloh and de Mare's accumulator can be compromised if an attacker can choose a subset of the values being accumulated (Fazio and Nicolosi, 2002). (Barić and Pfitzmann, 1997) proposed "Collision-free accumulators" to address this issue.

2.2.2 Collision-free Accumulator

The One-Way accumulators are elementary and ideal for Set Membership, whereas collision-free accumulators are more general (Barić and Pfitzmann, 1997), and are better suited for designing Fail-Stop signature schemes and Group Signatures. Collision-free accumulators consist of an accumulator scheme that is defined together by 4 polynomial-time algorithms: Gen, Eval, Wit and Ver.

The key generation algorithm *Gen* is used to generate the necessary key for a desired size accumulator, which accepts a security parameter 1^λ and accumulator threshold value N. The threshold value defines the maximum number of elements that can be accumulated securely. Gen returns a key k from key space $K_{\lambda N}$. The evaluation algorithm $Eval$ is used to accumulate elements of set $A = y_1, \cdots, y_{N'}$ where $N' \leq N$. It accepts the accumulator key k and values to be accumulated, $y_1, \cdots, y_{N'}$, as input. Eval returns the accumulated value z and an auxiliary information, *aux* that will be used by other algorithms. It is important to note that the Eval algorithm must return the same z value for the same input, but it may generate different auxiliary information. The witness extraction algorithm Wit generates the witness for a given input. It accepts the accumulator key k, the input value $y_i \in Y_k$ and the previous auxiliary and accumulator value z generated by Eval. Wit returns a witness value $w_i \in W_k$ to show input y_i was accumulated. Otherwise it returns the symbol $\perp$. The verification algorithm *Ver* is used to test the existence of an element in an accumulator; it accepts the accumulator key k, accumulator z, input value y_i, and its corresponding witness w_i. Ver returns the value TRUE or FALSE.

The accumulator scheme is paired with the property of *collision freeness*. Collision freeness ensures that an adversary bounded by polynomial-time cannot generate a set of values $Y = y_1, y_2, \cdots, y_N$ that produces an accumulator value z that allows for a value $y \notin Y$ and a witness w that allows for y to be proven as a member accumulated in z.

(Barić and Pfitzmann, 1997) propose two theoretical constructions of collision-free accumulators for building Fail-Stop Signature schemes. There also exists a dynamic accumulator implementation by (Camenisch and Lysyanskaya, 2002) that was inspired by the Collision-Free accumulator architecture but meant for set-membership testing.

2.3 Properties

Cryptographic accumulator properties are discussed next.

2.3.1 Security Properties

There are four prominent security properties of asymmetric cryptographic accumulators: *soundness, completeness, undeniability and indistinguishability*.

Soundness (aka Collision-Freeness) is defined as the probability of computing a membership witness for an element that is not part of the accumulated set or non-membership witness for an element part of the accumulated set is negligible (Derler et al., 2015). *Completeness*, (aka Correctness), property requires that all honestly accumulated values be verified as true with their respective witnesses with a negligible probability of error (Derler et al., 2015). An accumulator is called *undeniable* if the probability of computing a membership and non-membership witnesses together, of the same input, is negligible. Undeniability implies the collision-freenes property but not all collision-free accumulators have the undeniability property (Derler et al., 2015). *Indistinguishability* is both a security- and privacy-related property. An accumulator is indistinguishable if no information about the accumulated set is leaked by either the accumulator or its witnesses. This can be achieved by either accumulating a random value from the accumulation domain or using a non-deterministic Eval Algorithm (Derler et al., 2015).

2.3.2 Optional Crypto-Accumulator Features

Constructions can be optimized by selecting a subset of available features. Each available feature comes with a cost that can significantly impact the system design and implementation. Notable features are discussed next.

The set of accumulated elements is not always static; new elements may be added to or removed from this set over time. This is a common case for applications requiring to grant / revoke the privilege of a credential If an accumulator only supports additions, it is termed *additive*. Similarly, an accumulator that only supports deletions is termed *subtractive*. Addition and subtraction can be performed by redoing the accumulation process after updating the set. But, this approach is generally not practical, since recalculation takes polynomial time and depends on the size of the accumulated set (Baldimtsi et al., 2017). Dynamic accumulators are those that can efficiently (in sublinear or constant time complexity) update the accumulator and the respective witness values when a new element is added to (resp, removed from) the accumulated set (Au et al., 2009). Accumulators that do not support additions or deletions are termed *static*.

Accumulators are alo termed *positive*, *negative*, or *universal* based on the type of membership proof(s) they can support (Baldimtsi et al., 2017). Positive accumulators only support set membership proof of inclusion. Thus, for all elements in the accumulated set, there exists an efficiently computable witness w. Negative accumulators can only support a non-membership proof. For all elements that are considered non-members in regards to the accumulated set, there exists an efficiently computable witness w'. Lastly, a universal accumulator supports both membership and non-membership proofs (Li et al., 2007).

Zero Knowledge Proof (ZKP) is a privacy preserving membership proof used by cryptographic accumulators. It was initially proposed by (Goldwasser et al., 1989) in 1989. With ZKP, it is possible to show the accuracy of a statement about a secret without revealing the secret itself. This is possible because, if one can compute the same output that the prover provides by only accessing the input of the verifier; it should also be possible to compute the output before such interaction occurs. Therefore, through ZKP systems, an honest verifier need not interact with the prover (Morais et al., 2018) to verify the accuracy of a statement.

A single accumulator data structure alone won't be enough to implement ZKPs but two or more cryptographic accumulator schemes can be used to implement ZKP system. Even with a combination of accumulators, the interaction mechanisms must be augmented to carry out ZKPs. Additionally, these interactions must be standardized to make all parties aware of these mechanisms.

The zero knowledge property has been proven to imply indistinguishability for cryptographic accumulators (Ghosh et al., 2016). But, maintaining the integrity, zero knowledge property, and the efficiency of the accumulator simultaneously is challenging. This complexity can be reduced by restricting the accumulator's design to a trusted setup (Ghosh et al., 2016), where the accumulator value is always maintained by a trusted party. But, a trusted user is required to generate a secret value called a *trapdoor* to compute the accumulator and witness values efficiently. This setup raises major security and centralization concerns. But, there is no practical trapdoorless (strong) accumulator that can produce constant-size proofs. In trapdoorless accumulators, since the accumulator manager is also considered untrusted, the witness size grows at least like $\log N$ where N is the number of accumulated elements (Ghosh et al., 2016).

Applications can be either local, where only a single entity / authority is responsible for proving membership, or distributed where multiple parties in a network interact with each other. In the distributed case, the communication channel between all relevant parties becomes a bottleneck because membership witness additions or updates must be broadcasted each time an element is added to or deleted from the accumulator. To minimize this communication overhead, asynchronous accumulators were proposed for distributed applications (Reyzin and Yakoubov, 2016).

The asynchronous accumulator relies on low update frequency and compatibility with old accumulator version (Reyzin and Yakoubov, 2016). An accumulator has a low update frequency property, if the witness of an element requires less number of updates than the number of elements added after the element (Reyzin and Yakoubov, 2016). Low update frequency can be achieved only if it is additionally possible to accurately verify membership using an outdated accumulator value, in other words, the accumulator must be backwards compatible. It is important to note that the element tested for membership proof must be accumulated before the outdated accumulator value was generated (Reyzin and Yakoubov, 2016).

3 COST-BENEFIT ANALYSIS

Security properties of cryptographic accumulators define the reliability of the accumulator design. It is expected for an accumulator to satisfy these properties. However, some of the features presented in Section 2.3.2 are design choices rather than a necessity. We encountered four major design choices in literature and each choice affects the implementation complexity and overall accumulator performance. In this section we will present and discuss effects of these design choices in terms of space complexity, time complexity, and communication overhead.

In an application, the input set that needs to be accumulated can be static, additive, subtractive or dynamic. A static set is a set whose list of elements do not change through time. In an additive and subtractive sets, elements can only be added to or deleted from the set respectively. A dynamic set refers to the ability to both add and delete elements from the set. In our discussion, static accumulators will be our baseline and the system's initial space, time complexities and communication overhead will not be considered.

The effects of dynamic input set over system space, time complexity and communication overhead is presented in Table 1. This table is compiled considering major asymmetric cryptographic accumulators

Table 1: The space-, time-complexity and communication overhead in accumulators with input set changes. Independent cases separated using vertical line (|) and mutually exclusive cases separated using slash symbol (/).

	Additive	Subtractive	Dynamic
Space Complexity ($Size_{Acc} \mid Size_{Wit} \mid Size_{AM}$)	$O(1) \mid O(1) \mid O(1)$	$O(1) \mid O(1) \mid O(N)$	$O(1) \mid O(1) \mid O(N)$
Time Complexity at AM ($Wit_{Member} \mid Wit_{Non-member}$)	$O(1) \mid O(N)$	$O(N) \mid O(N)$	$O(1) \mid O(N) \mid O(N)$
Communication Overhead At AM	$O(1)$	$O(N)$	$O(1) \mid O(N)$

with the exception of the tree based strong accumulator, Merkle Tree. More information about Merkle Tree based accumulators can be found in the discussion about the accumulator manager (AM) trust.

When we evaluate space complexity based on a dynamic input set, we will investigate accumulator, witness and accumulator manager storage size changes (see Table 1). Size of an asymmetric accumulator can be specified using an accumulator threshold value N during accumulator generation. This value defines the upper bound on the total number of values that can be accumulated securely in the accumulator (Fazio and Nicolosi, 2002). Adding and removing elements to/from these accumulators do not affect the accumulator and witness size. On the other hand, RSA and Bilinear Map based accumulators require a trapdoor upon the initialization of the accumulators themselves as well as during the deletion of a member in the accumulated set. In otherwise trapdoorless constructions, accumulators' manager storage can grow proportional to the set size.

For every element added to the set, the accumulator manager must update the accumulator value and create a membership witness. The accumulator manager must also update the accumulator value after each deletion of an element and provide a non-membership witness if supported by the accumulator construction. Accumulator update upon element addition and subtraction and generating a membership witness for a new element are constant time processes done by the accumulator manager. On the other hand, generating a non-membership witness for RSA and Bilinear Map accumulators requires more time proportional to the number of elements in the set.

Asymmetric accumulators provide a constant verification and witness update time in case of an addition and subtraction of an element. However, the witness update process may cause a bottleneck in RSA and Bilinear Map accumulators during element subtraction since the witnesses have to be updated by the accumulator manager itself. However, there are proposed constructions in the literature that allow for the accumulated members to update their own witnesses after deletion by using Bezout coefficients (Baldimtsi et al., 2017; Li et al., 2007).

Table 2: The space-, time-complexity and communication overhead experienced by accumulator managers (AM) with different membership proof types. Symbols a, d, and S $(a-d)$ denote the number of elements added, deleted, and present respectively. '$\sim$' represents the range.

AM Complexity/Work	Positive	Negative	Universal
Space Complexity	$O(1) \sim O(d)$	$O(S)$	$O(S) \sim O(a)$
Time Complexity	$O(1)$	$O(1)$	$O(1) \sim O(S)$
Communication Overhead	$O(d) \sim O(a+d)$	$O(a+d)$	$O(a+d)$

In synchronous accumulators, the AMs must broadcast information about new elements to witness holders. This communication overhead is minimized in asynchronous accumulators, since witness holders need not update accumulator/witness value as frequently. But, RSA and bilinear-map accumulators experience additional communication complexity after element subtraction because of the queries from witness holders needed to retrieve updated witnesses from the accumulator manager.

Cryptographic accumulators can be positive, negative, or universal based on the membership proof they provide. Positive Accumulators provide only membership proofs to determine whether an element has been accumulated to the set of members. Negative accumulators, on the other hand, provide only non-membership proofs to determine whether an element has been accumulated to the set of non-members. Universal accumulators can provide both membership and non-membership proofs. Table 2 describes the time complexity, space complexity, and communication overhead of an accumulator based on the (non-)membership proof. They show the minimum and maximum values observed across all membership proof types. To identify generalized operations found across a class of accumulators with similar applications, we limited the scope by selecting dynamic accumulators that require a trusted setup with an accumulator manager (AM) and synchronized communication. Also, the values for the Merkle Tree accumulator were omitted from Table 2; they will be discussed in the evaluation of Table 3.

The values in Table 2 convey the range of performance, including bottlenecks, one may achieve by utilizing positive, negative, or universal accumulators. The Universal Accumulator has the highest complexity in all three dimensions when compared with (Strictly) Positive or (Strictly) Negative accumulators. However, the time complexity in Universal Accumulators may be reduced by implementing a separate accumulator of positive and negative types.

The space complexity for applying positive cryptographic accumulators is of the order of $O(1)$ for positive accumulators with the exception of BraavosB's (Baldimtsi et al., 2017) construction, due

Table 3: The space-, time-complexity and communication overhead in accumulators with Trusted AM and Untrusted AM. Symbols a, d, and S $(a-d)$ denote the number of elements added, deleted, and present respectively.

	Trusted AM	Untrusted (Strong) AM
Space Complexity ($Size_{Acc}\|Size_{Wit}\|Size_{AM}$)	$O(1) \mid O(1) \mid O(S)$	$O(1) \mid O(\log a) \mid O(a)$
Time Complexity (For all operations)	$O(1)$	$O(\log a)$
Communication Overhead	$O(a+d)$	$O((a+d)\log a)$

to its reliance on the Range-RSA construction for accumulating deleted elements. For negative and universal accumulators, space complexity is generally $O(S)$, which is the number of elements that are part of the accumulator at any given moment (not including elements that were deleted).

The computation time complexities mentioned in Table 2 show that all operations are typically of $O(1)$ except the NonMemWitCreate operation, which is the most expensive operation if using Universal accumulators built upon RSA and Bilinear Mapping cryptographic algorithms.

The communication overhead is $O(a+d)$ for dynamic accumulators, regardless of whether they are positive or negative or universal, due to communication being required for Member Witness updates after both additions and deletions of elements. For positive accumulators like CL-RSA-B, Braavos or BraavosB that are custom designed to minimize communication, the communication overhead is $O(d)$ because they don't require communication after addition of elements but only after deletion of elements.

Based on system requirements, a cryptographic accumulator can be designed either on the foundation of a trusted or untrusted accumulator manager. The RSA and bilinear map implementations are restricted to only trusted accumulator manager setups as a result of their reliance on a secret trapdoor value for initialization and constant time operations. The only known family of accumulators that can be safely used in an untrusted system are those based on Merkle Trees. This is because Merkle Trees do not require trapdoor values at any point during their operations.

Trusted accumulator manager systems use accumulators that do not provide the strong accumulator property. Accumulators based on RSA and bilinear maps fall into these types of accumulators. As shown in Table 3, trusted systems are relatively more scalable with $O(1)$ time and space complexities. And, similar to systems without trusted AM, the communication overhead grows linearly with the number of member additions and deletions. However, systems with trusted accumulator managers (AM) are centralized. The AM is required to maintain the integrity of the accumulator by safeguarding the trapdoor value. In case the trusted AM gets compromised, then the entire system may be brought down. Acquisition of the trapdoor value would allow an adversary to produce membership witnesses that are compatible with the accumulator value even for non-accumulated elements, threatening the *soundness* of the accumulator. This centralized infrastructure also limits the system's ability to distribute computational load, placing more responsibility on the trusted accumulator manager.

Untrusted accumulator manager (AM) systems use strong (trapdoorless) accumulators like Merkle Trees. Their operations require $O(\log a)$ time complexity; they produce witnesses of size $O(\log a)$, and their communication overhead is of superlinear time complexity of $O((a+d)\log a)$. Systems with untrusted AMs require relatively more space and can be expensive to maintain. The size of a Merkle Tree scales like $O(a)$ and does not reduce in size after deletions. Thus, the Merkle Tree will continue to grow regardless of the number of deletions.

Advanced security is the main advantage of untrusted accumulator manager systems. The use of a trapdoor is not required in a strong accumulator, and the responsibility of maintaining the integrity of the accumulated set is distributed amongst all participating untrusted accumulator managers. Further, he absence of a trapdoor allows for distribution of the computational load among the untrusted managers and accumulated members.

An asynchronous accumulator must hold both the low-update-frequency property and backwards compatibility property. To the best of our knowledge, the only construction that provides a fully asynchronous accumulator was defined by (Reyzin and Yakoubov, 2016). Another asymmetric accumulator that exhibits the low-update-frequency property but not the old accumulator compatibility property is the CL-RSA-B accumulator, presented by (Camenisch and Lysyanskaya, 2002). We refer to accumulators that only hold the low-update-frequency property as *partially asynchronous*. These accumulators are notably only positive accumulators. But, we are not claiming that the construction of a negative or universal asynchronous accumulator is infeasible; this remains an open question.

The asynchronous accumulator defined by (Reyzin and Yakoubov, 2016) takes the form of a dynamic set of Merkle Trees. The number of Merkle Trees grows by a factor of $D = log_2(n+1)$. Each subsequent Merkle Tree holds varying numbers of members and the entire construction follows a combinatorial approach when additions are made (Reyzin and Yakoubov, 2016). The benefit of using this

Table 4: The space-, time-complexity and communication overhead in accumulators with varying frequency in updates. Symbols a, d, and S $(a-d)$ denote the number of elements added, deleted, and present respectively. '$\sim$' represents the range

	Partially Asynchronous (Trusted)	**Asynchronous (Untrusted)**	**Synchronous (Trusted & Untrusted)**
Space Complexity ($Size_{Acc}\|Size_{Wit}\|Size_{AM}$)	$O(1)$ | $O(1)$ | $O(1)$	$O(\log a)$ | $O(\log a)$ | $O(a)$	$O(1)$ | $O(1) \sim O(\log a)$ | $O(S) \sim O(a)$
Time Complexity (For all operations)	$O(1)$	$O(\log a)$	$O(1) \sim \log a$
Communication Overhead	$O(d)$	$O(\log a + \log d)$	$O(a+d) \sim O((a+d)\log a)$

accumulator is the reduced communication overhead by a factor of $O(\log a + \log b)$, as shown in Table 4. The asynchronous accumulator can also be applied in decentralized, untrusted systems because of its strong accumulator construction. The disadvantages are the dynamic sizes of the accumulator value and witnesses by a factor of $O(\log a)$. The accumulator manager (AM) also requires storage of $O(a)$. Verification takes $O(\log a)$ as well. To support deletion, a list of all member values must also be stored and maintained (Reyzin and Yakoubov, 2016).

The partially asynchronous variant in the form of the CL-RSA-B accumulator has the advantage of constant time operations and constant size accumulator and witness values. It also only has to update witnesses after a member is deleted from the set; witness updates are not required after additions. For an accumulator scheme that is not expecting a significant number of deletions compared to additions, this feature is ideal because the communication overhead is only d as shown in Table 4. The main drawback of a partially asynchronous accumulator is its inability to exhibit the old accumulator compatibility property, so the most up-to-date accumulator value must be held. Another disadvantage is its restriction to trusted accumulator schemes in which the trusted accumulator manager is the only entity that can generate witnesses upon the addition of new members and update the accumulator value when a member is deleted. This is due to the requirement of a trapdoor to perform deletions in a CL-RSA-B accumulator. Once a member is deleted, the new accumulator value as well as the deleted member's ID is broadcasted to the other members to update their respective witness values (Baldimtsi et al., 2017).

In comparison with synchronous accumulators, overall system's communication cost is reduced in both asynchronous and partially asynchronous accumulators. Synchronous accumulators never have a communication overhead less than $O(a+b)$. This is because all witnesses must be updated after an member addition or deletion in the set and witnesses are incompatible with old accumulator values. The decision of which asynchronous accumulator type to pick comes down to whether the system requires a trusted or untrusted setup. Dynamic or static sizes of witness and accumulator values should be taken into consideration too.

4 CONCLUSION

We provided a concise guide on cryptographic accumulators. We presented their security and optional properties. Also, we discussed the effects of different optional properties on accumulator performance in terms of space, time, and communication complexity.

Cryptocurrencies were early adopters of cryptographic accumulators. Bitcoin uses a Bloom Filter for set-membership testing of transactions (Bitcoin.org, 2020a) . Bitcoin also uses Merkle Block (Bitcoin.org, 2020b) to confirm the validity of transactions without having to download and/or maintain the entire copy of blockchain by every participating node. In Zerocoin (Miers et al., 2013), a CL-RSA-B based accumulator is implemented to represent a set of all transactions that have been committed to blockchain (Miers et al., 2013). The accumulator additionally eliminates trackable linkage of addresses and facilitates anonymous and untraceable public transactions (Miers et al., 2013).

Other applications of crypto accumulators include, but are not limited to, maintaining a certificate revocation list, blacklisting or whitelisting user credentials in an authentication system and generating important client groups such as lists of high risk and/or bankrupt clients

Cryptographic accumulators can also help maintain dataset privacy while sharing. They can be implemented to share and maintain a list of such users without actually revealing their identities. This property could simplify sharing critical data sharing with third parties. Cryptographic accumulators can also be used in offline ID verification, credit score checking, and medical-data verification.

ACKNOWLEDGMENTS

This material is based upon work supported by, or in part by,the National Science Foundation (NSF) under grants CCF-1562659, CCF-1562306, CCF-1617690, CCF-1822191, CCF-1821431, and The Scientific and Technological Research Council of Turkey (TUBITAK). The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, views, or endorsements, either expressed or implied, of the NSF or TUBITAK.

REFERENCES

Akkaya, K. and Cebe, M. (2018). Efficient public-key revocation management for secure smart meter communications using one-way cryptographic accumulators. In *IEEE International Conference on Communications*. Florida International University.

Au, M. H., Tsang, P. P., Susilo, W., and Mu, Y. (2009). Dynamic universal accumulators for ddh groups and their application to attribute-based anonymous credential systems. In *Cryptographers' track at the RSA conference*, pages 295–308. Springer.

Baldimtsi, F., Camenisch, J., Dubovitskaya, M., Lysyanskaya, A., Reyzin, L., Samelin, K., and Yakoubov, S. (2017). Accumulators with applications to anonymity-preserving revocation. In *2017 IEEE European Symp. on Security & Privacy (EuroS&P)*, pages 301–315. IEEE.

Barić, N. and Pfitzmann, B. (1997). Collision-free accumulators and fail-stop signature schemes without trees. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 480–494. Springer.

Benaloh, J. and De Mare, M. (1993). One-way accumulators: A decentralized alternative to digital signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 274–285. Springer.

Bitcoin.org (2020a). Application of Bloom filters. https://bitcoin.org/en/operating-modes-guide#application-of-bloom-filters. [Online; Accessed:March/2020].

Bitcoin.org (2020b). Merkleblock. https://bitcoin.org/en/developer-reference#merkleblock. [Online; Accessed:March/2020].

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Comm. of the ACM*, 13(7):422–426.

Camenisch, J. and Lysyanskaya, A. (2002). Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Annual International Cryptology Conference*, pages 61–76. Springer.

Derler, D., Hanser, C., and Slamanig, D. (2015). Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *Cryptographers' track—RSA conf.*, pages 127–144. Springer.

Fan, B., Andersen, D. G., Kaminsky, M., and Mitzenmacher, M. D. (2014). Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 75–88.

Fazio, N. and Nicolosi, A. (2002). Cryptographic accumulators: Definitions, constructions and applications. *Paper written for course at New York University: www. cs. nyu. edu/nicolosi/papers/accumulators. pdf*.

Ghosh, E., Ohrimenko, O., Papadopoulos, D., Tamassia, R., and Triandopoulos, N. (2016). Zero-knowledge accumulators and set algebra. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 67–100. Springer.

Ghosh, E., Ohrimenko, O., and Tamassia, R. (2014). Verifiable member and order queries on a list in zero-knowledge. *arXiv preprint arXiv:1408.3843*.

Goldwasser, S., Micali, S., and Rackoff, C. (1989). The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208.

Goodrich, M. T., Tamassia, R., and Hasić, J. (2002). An efficient dynamic and distributed cryptographic accumulator. In *Int. Conf. on Info. Security*, pages 372–388. Springer.

Kumar, A., Lafourcade, P., and Lauradoux, C. (2014). Performances of cryptographic accumulators. In *39th Annual IEEE Conference on Local Computer Networks*, pages 366–369. IEEE.

Li, J., Li, N., and Xue, R. (2007). Universal accumulators with efficient nonmembership proofs. In *International Conference on Applied Cryptography and Network Security*, pages 253–269. Springer.

Merkle, R. C. (1989). A certified digital signature. In *Conf. on the Theory and Application of Cryptology*, pages 218–238. Springer.

Miers, I., Garman, C., Green, M., and Rubin, A. D. (2013). Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411. IEEE.

Morais, E., van Wijk, C., and Koens, T. (2018). Zero knowledge set membership.

Reyzin, L. and Yakoubov, S. (2016). Efficient asynchronous accumulators for distributed pki. In *International Conference on Security and Cryptography for Networks*, pages 292–309. Springer.

Slamanig, D. (2012). Dynamic accumulator based discretionary access control for outsourced storage with unlinkable access. In *Int. Conf. on Financial Crypto. & Data Security*, pages 215–222. Springer.

Sudarsono, A., Nakanishi, T., and Funabiki, N. (2011). Efficient proofs of attributes in pairing-based anonymous credential system. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 246–263. Springer.

Tremel, E. (2013). Real-world performance of cryptographic accumulators. Thesis, Brown U.

Weisstein, E. W. One-way function. from Mathworld - a Wolfram web resource. https://mathworld.wolfram.com/One-WayFunction.html. [Online; Accessed: March/2020].

Weisstein, E. W. One-way hash function. from mathworld - a wolfram web resource. https://mathworld.wolfram.com/One-WayHashFunction.html. [Online; Accessed: March/2020].