Securing Fieldable Bioinformatics

Daniel Wyschogrod *Raytheon BBN* Cambridge, MA, USA dan.wyschogrod@rtx.com Steven T. Murphy *Raytheon BBN* Cambridge, MA, USA steven.t.murphy@rtx.com Jacob Beal *Raytheon BBN* Cambridge, MA, USA jakebeal@ieee.org Allison Taggart Raytheon BBN Cambridge, MA, USA allison.j.taggart@rtx.com

Abstract-Nanopore sequencing offers promising potential for rapid and target-agnostic sensing and diagnostics in field applications. However, realizing this potential necessitates addressing not only sample collection and processing challenges but also concerns related to information security and privacy when deployed in field devices. We thus introduce the Secure Bloom-Filter Analysis and Compression (SB-FAC) architecture, which breaks bioinformatics computations into a Bloom-filter-based field preprocessing stage that identifies regions of interest in the raw read data and a server-side interpretation stage that combines and interprets these identified regions. Sensitive information encoded in the Bloom filter can be protected from extraction by the use of a cryptographic hash paired with a salt in a Trusted Platform Module (TPM). We experimentally validate the predicted scaling of this approach, confirming that cost per operation linear in salt length, while reverse engineering cost is exponential.

Index Terms-Bioinformatics; Bloom filter; Security

I. INTRODUCTION

Nucleic acid sequencing methods have advanced rapidly in recent years, becoming faster, more robust, and more affordable. Nanopore sequencing, in particular, offers a combination of high speed and low resource requirements that make it attractive for a variety of point-of-need applications, such as environmental sensing, disease surveillance, and diagnostic testing [1], [2]. These devices are particularly valuable for conducting untargeted genomics or transcriptomics, which can allow a fielded capability to be reconfigured, e.g., for emerging pathogens or new biomarkers of interest, without the need for new reagents. Realizing this potential will require addressing a number of well-understood challenges related to sample collection and processing, including sample preparation [3], error rate [2], and bioinformatic processing [1].

For bioinformatic processing, the substantial data volumes generated by nanopore sequencing present challenges regarding the processing location. These computations often involve sensitive information, pertaining to biosecurity or privacy concerns. While cryptographic computational techniques, such



Fig. 1. Secure Bloom-Filter Analysis and Compression (SB-FAC) architecture: sequencing data is preprocessed in the field using a secure Bloom filter to identify data of interest, and only this filtered subset is related to the data center for interpretation.

as homomorphic encryption [4], [5], offer secure processing at the data generation point, they are often prohibitively computationally intensive, rendering them impractical for most bioinformatic applications. Alternatively, if bioinformatics computations are not performed in the field, significant hurdles arise due to the necessity of transferring large volumes of sequencing data from the generation site to a secure data center.

Here we propose an alternative approach, the Secure Bloom-Filter Analysis and Compression (SB-FAC) architecture (Figure 1), which aims to allow secure fieldable bioinformatics by encoding bioinformatics functions as Bloom filters [6] in a manner that is difficult to reverse engineer. In Section II, we describe the SB-FAC architecture and its relationship to prior work. Section III analyzes potential security concerns, identifying protection of the contents of the Bloom filter as the key challenge not already addressed by conventional security methods. Section IV then considers potential mitigation strategies and their relative costs and benefits. From this, we select an approach of cryptographic hashing with an added salt value using a Trusted Platform Module (TPM), and analyze its scaling properties in Section V, which are then verified experimentally in Section VI. Finally, Section VII summarizes contributions and potential directions for future work.

This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), and the Naval Information Warfare Center - Pacific (NIWC-P) via contract N66001-23-C-4502. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein. This document does not contain technology or technical data controlled under either the U.S. International Traffic in Arms Regulations or the U.S. Export Administration Regulations.

II. BLOOM-FILTER FACTORING OF BIOINFORMATICS

A Bloom filter [6] is a data structure that uses hash functions for fast probabilistic testing for whether a query item is a member of a set. Set members are encoded by setting bits at locations determined by multiple hash functions; conversely, a query item is tested for membership by checking to see whether all of the bit locations determined by the hash functions have been set. If any bit location that is hashed to is not set, this is a negative response that indicates the query item is definitely not part of the set. If all of the bit locations are set, on the other hand, this indicates either that the query item is part of the set or else a false positive due to hash collisions, where the probability of a false positive is determined by the size of the Bloom filter, the number of hash functions, and the density of bits that are set. Many variants of this database structure exist, including Counting Bloom Filters [7], Invertible Bloom Filters [8], and Approximate State Machines [9], each with its own strengths and weaknesses.

Mathematically, every function can in principle be transformed into a sequence of tests for set membership, but such a transformation is only practical for some functions. In recent years, a number of bioinformatic tools based on Bloom filters have been developed (e.g., [10]–[15]), along with applications to a wide range of bioinformatics applications (e.g., [16]–[21]).

In our own prior work, the FAST-NA biosecurity system [22], [23] breaks the challenge of identifying controlled pathogens or toxins into two stages. First, a pre-processing stage uses a Bloom filter to rapidly and cheaply identify all k-mer fragments of a nucleic acid sequence that may belong to controlled pathogens or toxins. Second, an interpretation stage takes this much smaller stream of fragments and removes false positives, then reconstructs and interprets regions of interest. This Bloom filter architecture is fast enough to keep up with a MinION sequencer in real-time [24].

The Secure Bloom-Filter Analysis and Compression (SB-FAC) architecture (Figure 1) derives from the insights that: 1) these two stages need not run on the same machine, 2) many other bioinformatic functions can similarly be factored into a "high data, low knowledge" filtering stage followed by a "low data, high knowledge" interpretation stage, and 3) Bloom filters are, by their nature, inherently opaque and difficult to reverse engineer.

Many bioinformatic computations that are useful for field applications can be readily factored to map into the field and server components of the SB-FAC architecture following two general patterns, matching and subtraction (Figure 2). For matching functions, a Bloom filter is constructed using kmers unique to sequences of interest. Input sequencing data is then broken down into k-mers and compared to the Bloom filter. A Boolean value of present or absent is determined for each k-mer, generating a list of matching k-mers. Either just these k-mers or any read containing at least one matching k-mer are then sent to the data center, where the matches are validated by comparison to the original sequences of interest, then further processed to provide summary results



Fig. 2. Two useful patterns for implementing bioinformatics functions via Bloom filters are matching (a), in which computations are performed on sequences that match against a Bloom filter encoding a database of signatures, and subtraction (b), in which computations are performed on sequences that do not match a Bloom filter encoding a database of background materials to be removed.

for interpretation. Examples of matching functions include screening for pathogens or toxins, detection of engineering markers, tests for health-related biomarkers, and taxonomy identification (e.g., by rRNA matching).

Subtraction functions are the converse: a Bloom filter is constructed using k-mers that identify sequences that are not of interest, such as host DNA in a diagnostic sample. As with matching, input sequencing data is broken into k-mers and tested for presence or absence in the Bloom filter. In this case, however, it is the matching k-mers that are removed, and the non-matching k-mer sequences or reads containing them that are sent to the data center for validation and further processing. Examples of subtraction functions include contamination filtering, variant calling, and identification of novel sequence material. Finally, we note that some functions may use both matching and subtraction, which can be implemented simply by using two Bloom filters. For example, a diagnostic process might want to both remove host sequence and perform taxonomic identification on the remaining material.

III. SECURITY CONCERNS

In terms of security analysis for Bloom filters [9], various metrics are employed. In a deployed system, an adversary gaining physical access to the hardware might have access to both the code and the Bloom filter, which differs from certain privacy analyses presuming the Bloom filter is secured within a server [10].

Given such access, an adversary could attempt to extract information from the Bloom filter. This extraction could involve estimating the number of entries by examining the fraction of set bits and testing for the presence of specific k-mers. However, both of these properties can be made more obscure in our scenario through specific measures (outlined below).

It is important to note that if cryptographic hash functions are employed, the only feasible method for an adversary to



Fig. 3. Information flow for SB-FAC, marking key vulnerabilities in red.

extract the complete set stored in a Bloom filter is through brute force [9]. This is due to cryptographic hash functions possessing crucial properties, such as pre-image resistance attained through the uniform distribution of results.

The goals of the attacker will depend on the application and the manner in which the SB-FAC system is being employed. Applications include but are not limited to:

- Set membership
- · Identification of controlled pathogens and toxins
- Taxonomic classification
- Identification of engineered sequences
- Health biomarkers
- Extract novel sequences

At a high level, attackers goals may include:

- Learning about the mission being conducted
- Identifying mission goals and targets
- Interpreting results obtained during the mission
- Learning about the bioinformatics function implemented

We consider attackers with various levels of access to the SB-FAC system, sorted from least to most:

- Attacker has access to the field environment (premission, during mission, and/or post-mission), allowing them to evaluate and/or disperse environmental biological material and monitor wireless transmissions
- Attacker has access to the fieldable device (postmission), adding the ability to input samples with known content and monitor device activity level (e.g., power consumption)
- Attacker has access to SB-FAC API (post-mission), adding the ability to input specific sequences and receive unencrypted device outputs
- 4) Attacker has access to SB-FAC source code and data (post-mission), providing knowledge of all functionality not encoded in Bloom filter with the ability to test for inclusion of specific sequences in Bloom filter

Note that normal physical security methods can attempt to prevent level 3-4 access, but these are imperfect, so we will assume level 3-4 access is possible for a highly capable attacker.

For many of the above applications, the specific technical goals include:

- Capturing the contents of the samples being tested (Goal 1, 3)
- Finding out as large a portion of the k-mers contained in the Bloom filter as possible (Goal 2,3,4)
- Checking whether k-mers of concern to the attacker are contained in the Bloom filter (Goal 2)
- Determining what matches are found for a particular sample (Goal 1, 2, 3)
- Injecting either positive or negative samples into the input stream (Goal 2)
- Forcing the system to take unwanted actions (e.g. alert on a negative sample, deny a synthesizer permission to synthesize a sequence, etc.) (Goal 2b)
- Force a misclassification of a sample (Goal 2b)

A. Analysis of Information Flows and Vulnerabilities

The diagram above (Figure 3) is a block diagram of an SB-FAC system from a security perspective. At the highest level, it shows that the system runs on two hosts (these may be containers, VMs, or other instances). The first machine is the client, which is distributed to users and is thus easier to compromise since every user has a copy of the data and executable. Therefore, the goal is to both make its operations as minimal as possible and for it to expose minimal information. Maximal intelligence is placed on the server to extract as much as possible from the results sent by the client.

At the next level, we analyze the inputs and outputs of the two machines. The field device takes two inputs and has only one output. The inputs and output for the field device are:

Bloom Filter - The server generates a distinct per-client Bloom filter, incorporating keys and potentially employing unique watermarking specific to each client. While multiple clients could theoretically receive identical Bloom filters in terms of functionality, creating individualized Bloom filters offers several advantages. These include the implementation of per-user watermarking to prevent and trace data exfiltration, as well as the utilization of customer-specific keys to obscure data.

Under normal security measures, access to this input is restricted for attacker access levels 1-3. However, at attacker access level 4, complete access to the contents of the Bloom filter becomes available.

Input Sequence Data - The input data, encompassing biological sequences like nucleic acids or proteins, is user-provided. Attacker access levels determine the extent of control over input samples:

- Level 1: Potential acquisition of sample distribution through independent environmental sampling.
- Level 2: Indirect control via providing known-content biological material.
- Levels 3-4: Full direct control over input samples, enabling comprehensive manipulation.

Filtered Sequencing Data - This is the output from the Bloom filter, possibly consisting of raw match sequences and offsets, possibly post-processed. An attacker with access level 1-2 can observe traffic patterns. An attacker with access level 3 has

complete observation of output match results. An attacker with access level 4 has complete access to raw match results and their transformation for transmission.

Given the levels of access, relative to the attacker goals above, we can see that most of the system and its operations can be effectively defended with standard network and datacenter security methods. The critical elements of the fielded system that need additional consideration are the output matches and the Bloom filter itself.

IV. MITIGATION STRATEGIES

Taking into account the analysis of attack types and the crucial data requiring protection, we explored diverse defense strategies. Initially, we address methods safeguarding the match output, followed by an exploration of mechanisms aimed at defending the Bloom filter's contents.

A. Increase False Positive Rate

In the face of brute force attacks, increasing the number of false positives in the match stream requires the attacker to filter through all positives emitted by the system to find the true positives. This makes the attacker's task more difficult whether they can only see match results, results and sample data, data from multiple clients, or even when they have access to the Bloom filter and are trying brute force searches on their own. All Bloom filters have false positives, but the rates can be set to be extremely low, and this is typically the preferred mode of operation. The obscuration advantage of the false positive rate thus needs to be weighed against the increase in field system to server traffic. Below we discuss various mechanisms for increasing false positive rates.

1) Setting Random Bits: If additional bits are randomly set in the Bloom filter, the number of false positives will increase. Per [25], the false positive rate for a Bloom filter is $f = (b/m)^k$, where f is the false positive rate, b is the number of bits on in the Bloom filter, and m is the total number of bits in the Bloom filter. For example, with 17 hash functions (the number used in FAST-NA [22], [23]) and 40% of the bits on, the false positive rate is $f = 1.7 \times 10^{-7}$. If random bits area added until the fraction on is raised to 50% of the bits, then the false positive rate rises to $f = 7.6 \times 10^{-6}$, a factor of about 44x.

2) Reducing Number of Hash Functions: As an alternative to adding random bits, the false positive rate can also be increased by reducing the number of hash functions k. For example, k is reduced from 17 down to 10, then at 40% of bits set that false positive rate rises to $f = 1.0 \times 10^{-4}$, a factor of about 610x. However, one needs a sharp reduction in number of hash functions to achieve significantly higher FP rates.

3) Using Decoy Hash Functions: Another approach, based on adding rather than removing hash functions, can add far more false positives without actually increasing the number of bits that are set in the Bloom filter. For this approach, instead of using k hash functions, we use k+d hash functions, letting d be the number of decoy hashes, but set bits in the Bloom filter at a randomly chosen subset of only k of the locations. To query items in the Bloom filter, locations are tested using all k + d hash functions, and a match is reported whenever at least k of the k + d bits are set. The number of combinations that will give a positive result is determined by the binomial coefficient, thus increasing the false positive rate by that factor without turning on any more bits would be used with only k hash functions. For example using k = 17 and d = 3, for a total of 20 hash functions, the apparent false positive rate would be 1.9×10^{-4} and 4.47×10^{-3} , orders of magnitude higher than by adding bits or reducing the number of hash functions.

4) Adding Structured Decoy K-mers: Finally, it is important to note that all of the above techniques will produce random false positives. Since actual sequencing data is structured, however, genuine hits will often come in clusters as a matched region of size n > k produces a sequence of n - k + 1overlapping k-mer matches. An attacker could thus discard matches that are isolated and do not overlap with other nearby matches. To address this issue, it is necessary to incorporate structured decoy material into the Bloom filter. This material should execute an unrelated function, possessing an arbitrary and non-sensitive nature.

B. Cryptographic Hash Functions

Cryptographic hash functions have high uniformity, which makes them ideal for using in a Bloom filter and intractable to reverse. Their computational cost, however, is typically significantly higher than that of other hash function alternatives, which can make it costly to calculate large numbers of cryptographic hash values, e.g., k = 17 for FAST-NA [22], [23]. A result from Kirsch & Mitzenmacher [26], [27], however, shots that given only two hash functions, an arbitrary number can be simulated using linead combinations of the form $g_i = h_1 + ih_2$, where h_1 and h_2 are the two independently calculated hash values. For SB-FAC, we can thus take two cryptographic hash functions (e.g., SHA-2 and SHA-3, or SHA-2 with two different prefixes) and turn them into all k of the hash functions required for a desired false positive rate.

C. Adding Secret Key Salt Prefixes or Hashes

To heighten the difficulty of reverse engineering the Bloom filter, one approach involves prefixing each string intended for addition to the Bloom filter with a confidential key (essentially acting as a salt) before the hashing process. This necessitates the client machine's awareness of the key to access a kmer, while emphasizing the need to safeguard this key from potential access by adversaries.

Our proposed solution involves encapsulating both the secret key and the execution of hash functions utilizing that key within specialized hardware, such as a Trusted Platform Module (TPM). The TPM, a chip or board designed for storing keys and other confidential data in a tamper-resistant manner, ensures enhanced security measures. Extracting keys from a TPM, even when an attacker has physical possession of the hardware, is widely recognized as an exceptionally challenging task.

V. ANALYSIS OF MITIGATION STRATEGIES

In this section, we analyze the expected efficacy of the combination of mitigation strategies, beginning by defining a worst case scenario, then analyzing the mitigation costs for both system and attacker in this scenario.

A. Worse-Case Scenario Security Metric

As mentioned previously, our analysis of potential security issues has identified that most of the potential security issues for the SB-FAC architecture can be addressed with existing methods, i.e., no research is necessary to address these issues. The key question remaining is whether an adversary can they be prevented from extracting potentially sensitive data that is encoded in the Bloom filter.

In general, the structured nature of biological sequence information means that the data stored in a Bloom filter is expected to have a significant degree of clustering. An attacker can thus potentially use domain knowledge to greatly reduce the amount of effort needed to extract the k-mers in a particular cluster, beginning with some form of "seed" based on known or predicted Bloom filter contents. An example is the overlap k-mer problem discussed above in Section IV-A4: once a single matching k-mer has been identified as belonging to the filter, it is likely that other overlapping k-mers are also included in the Bloom filter.

To measure our ability to defend against such attacks, we will thus make the following worst case assumptions:

- The adversary has complete access to the system, including the Bloom filter and the code, including the number and nature of hash functions for the Bloom filter, except for a single salt value that is *S* bits long, access to which is protected by a standard Trusted Platform Module (TPM).
- The adversary knows the length of the salt S, but not its value.
- The adversary can test for inclusion of k-mers in the Bloom filter.
- The adversary can use biological knowledge to identify some number of k-mers that must be stored in the Bloom filter.
- The false positive rate has not been enhanced (i.e., none of the methods in Section IV-A are in use).

The adversary can then extract information from the Bloom filter if they can identify the value of the salt. This value, in turn, can be brute forced by performing queries on k-mers known to be included in the Bloom filter until the adversary finds a salt value that consistently produces hits on included k-mers.

B. Numerical Analysis

Given these definitions, we define the system to be infeasible to reverse engineer if it has the following two scaling properties:

- Time per Bloom filter insertion or query T is at most linear in the length of a k-mer L plus the length of the salt, i.e., T = O(L + S)
- Identifying a salt value (thus allowing information extraction from the Bloom filter) requires an expected number of queries Q at least exponential in the length of the salt, i.e., Q = Ω(2^S).

If we can establish these two scaling properties, then there is little constraint on the salt length S and an SB-FAC system can be secured against reverse engineering by choosing S such that performing Q queries is far beyond the capabilities of any plausible attacker.

The first property, linear insertion or query, is trivial to establish: common cryptographic hash functions such as SHA-2 and SHA-3 all require at most linear time with respect to the string to be hashed, so the property T = O(L+S) can be assumed.

We now analyze the expected required number of queries Q to discover a salt. Given a Bloom filter with false positive rate F that is known to include a particular k-mer, testing all potential values for a salt with S bits requires $Q_1 = 2^S$ queries, where Q_1 is the number of queries for testing potential salt values for this first known k-mer. This set of queries is expected to produce one true salt value and $K_1 = QF$ false salt values.

To increase the likelihood of identifying the correct salt, the number of potential false salt values K_1 can be reduced by testing for additional known k-mers. For a second k-mer, only the $K_1 + 1$ salts from the first k-mer need to be checked, not the whole space of possible salts. For a third, only those salts that satisfy both of the first two, etc.

Given a uniform hash function, false salts can be assumed to be independent. This means that the number of potential false salt values reduces geometrically by F with each iteration, such that after X iterations the number of false salt values remaining to be eliminated is expected to be $K_X = Q_1(F^X)$, and that the number of queries required at each iteration is $Q_i = Q_1(F^{X-1}) = K_{i-1}$.

Let us consider the salt to be sufficiently well identified if there is at least a 50% chance of it being correctly identified. To solve for 50% certainty, then, we set $K_X = 1$, which implies:

$$\frac{1}{Q_1} = F^X$$
$$-log(Q_1) = X \cdot log(F)$$
$$X = -\frac{log(Q_1)}{log(F)}$$

The total number of queries Q to determine the salt is thus expected to be:

$$Q = \sum_{i=1}^{X} Q_i = Q_1 \sum_{i=0}^{X-1} F^i$$

By the sum of geometric series, the value of Q is thus bounded:

$$Q_1 < Q < Q_1 \frac{1}{(1-F)}$$

$$2^S < Q < \frac{2^S}{(1-F)}$$

For any practical Bloom filter, the value of F should be small, i.e., $F \ll 1$, and thus the expected number of queries required is $Q = \Theta(2^S)$, satisfying the required property.

Given this analysis, testing whether a given k-mer is stored in the Bloom filter equivalent to determining the salt for the Bloom filter. As such, extracting data from a Bloom filter is also at least as difficult as determining the salt, i.e., $Q = \Omega(2^S)$, since any given data extraction goal will involve testing the Bloom storage for at least one k-mer, and possibly many more.

Finally, we can ask what value of S is sufficient for strong protection. Here, we will compare against one of the current US government standards, the Advanced Encryption Standard (AES) algorithm. The current recommended distribution of AES in the Commercial National Security Algorithm Suite (CNSA) uses 256-bit keys. We can thus conclude that S = 256bits should be a sufficiently large salt for SB-FAC to prevent brute-forcing, at least matching this state-of-the-art encryption standard (and in fact, SB-FAC is likely to be more difficult to brute force due to false positives in the Bloom filter). Our prior work with Bloom filters has used k-mers that are strings of length 14 (112 bits) to 42 (336 bits) [22], [23], so given the linear scaling of insertion and query operations, the computational cost for using 256-bit salts is predicted to be only a modest increase, in the range of 1.8x - 3.3x, and might be significantly less for hash algorithms with a high constant overhead. Thus, SB-FAC should be able to use salts of sufficient length to provide security at a computational cost reasonable for fieldable hardware.

VI. EXPERIMENTAL VALIDATION

Having established that an appropriate choice of hash functions should in theory allow secure field operations of Bloom filters with reasonable overhead, we now validate those design decisions with empirical investigation of the uniformity of the selected hash functions and of cost scaling with respect to salt length. For these experiments, we used an implementation of the SB-FAC architecture written in Python.

A. Hash Function Uniformity

The resistance of a Bloom filter to reverse engineering is similar to the resistance of its core component, the set of hash functions used. The security of a hash function is frequently characterized by its *pre-image resistance [28]*. An important characteristic of pre-image resistance is the uniformity of the generated values. In terms of Bloom filters, this means that if we divide the regions of the Bloom filter into "buckets", each containing a small number of bits relative to the entire Bloom filter bit array, we can assess uniformity based on the distribution of number of bits on per bucket.

If a small number of insertions are made into the Bloom filter, most buckets will have either zero or one bits on. If large numbers of bits are on, most buckets will have close to m/N, where m is the number of bits set in the Bloom filter



Fig. 4. Comparison of experimental and theoretical results for Bloom filter uniformity for two classes of hash functions. (a) shows the results for the two cryptographic hash using K&M and (b) shows the results for the Murmur hash set. (c) shows a synthesized bias case where 1000 of the bins are zeroed out and each of their neighbors is or'ed with their content

and N is the number of buckets. If regions of the Bloom filter are under or over populated, this demonstrates a bias on the part of the set of Bloom filters which an attacker could use to reduce the search space when looking for an entry.

In order to validate our selection of hash functions used to build the Bloom filter, we analyzed the uniformity of bits for various numbers of insertions into small Bloom filters, comparing against a mathematical model of uniform bit distribution.

The model is derived from the balls dropped randomly into N buckets problem. The probability is given by the binomial distribution:

$$Pr(X = x) = \binom{k}{x} p^k \left(1 - p\right)^{k-x}$$

where k is the number of balls to insert, each Bloom filter insertion adding h balls, where h is the number of hash functions, x is the value of the number of balls in a bucket, and p is the probability of insertion into any bucket. If we assume equal probability of $\frac{1}{N}$ for all buckets, the equation becomes:

$$Pr(X = x) = \binom{k}{x} \frac{(N-1)^{k-x}}{N^k}$$

In order to get the number of bins with x balls in them, we simply multiply by N, giving:

$$N_{balls_in_bin}(x) = NPr(X = x)$$

Finally, the above model would be correct if we could always be sure that new balls would accumulate in a receiving bucket. But, of course, this is not true since some bits that we try to turn on will already be on and these collisions won't accumulate new "balls". We can correct for how often that will happen by reducing the effective number of balls that we insert. A well known result for approximating the probability of a random bit in a Bloom filter still being zero after ninsertions is given by [29]: $e^{-kn/m}$

But this is only the incremental probability when n insertions have already taken place. The amount of reduction (R)due to collisions as we fill up the Bloom filter from empty to *n*insertions is given by the integral of the above quantity:

$$R = \int_0^n e^{-kx/m} dx$$

and the reduced number of balls is given by:

$$k_{\text{effective}} = m(1 - e^{-kx/m})$$

We have experimented with two types of hashes:

- 1) Murmur hash, full set these are non-cryptographic hash functions where new hashes are produces as needed by using different seeds, using in our prior work [22], [23].
- 2) Two cryptographic hashes combined using the Kirsch & Mitzenmacher algorithm - here we use only two hash functions, SHA-256 and MD5, combined in a linear function as specified in [26], [27]

Figure 4 compares model predictions to experimental results from 10 independent trials on a Bloom filter sized to hold 24,000 items with false positive p = 0.0001 and 10% occupancy, segmented into 4096 bins of 128 bits each, for the Murmur hash (Figure 4a), the Kirsch & Mitzenmacher synthesis of cryptographic hashes (Figure 4b), and for a biased set constructed by setting 1000 bins to zero and or'ing their neighbors with their content. For both the Murmur hash and the Kirsch & Mitzenmacher synthesis of cryptographic hashes, the observed and theoretical values are relatively close, without a clear bias in distribution as is shown in the third



Fig. 5. Comparison of exponential cost to crack a salted Bloom filter and the linear cost to utilize insertion and query operations. (a) Exponential scaling

of number of queries required to identify a salt given that the attacker has knowledge of a subset of the Bloom filter data. (b) Linear scaling of the cost of inserting (write) and querying (read) values in a Bloom filter. Note that cryptographic hashing with 256-bit salts with the Kirsch & Mitzenmacher optimization is significantly faster than the unsalted Murmur hash. Both graphs show the average over ten runs per condition, plus/minus one standard deviation.

case. This criteria thus shows no significant different in preimage resistance between the two hash function alternatives investigated. It is possible however, that the linear combination of cryptographic hashes may have adverse effects on their uniformity, which is a question for further investigation in the future.

B. Cost Scaling With Salt Length

We next checked whether cost scaling for both brute force information extraction and normal operations matches the predictions in Section V. To evaluate the cost of brute force identification of a salt value, we ran trials with a range of salt lengths from 1 to 20 bits, in which salts were guessed sequentially until one allowed identification of known item in the Bloom filter. Figure 5(a) shows the result of 10 independent trials on a Bloom filter sized to hold 24,000 items with false positive p = 0.0001, each holding 10 k-mers of 28 base pairs each, one of which is known to attacker. As expected, the scaling is exponential, and closely hews to the specific value predicted in Section V.

To evaluate the cost of operations, we tested the Kirsch & Mitzenmacher synthesis of cryptographic hashes with salt lengths ranging from 128 to 8192 bits, as well as the unsalted Murmur hash function using in prior work. Figure 5(b) shows the result of 10 independent trials on a Bloom filter sized to hold 10^6 items with false positive p = 0.0001, first writing 10^6 random sequences, followed by 10^6 reads for the same set of sequences. As expected, both insertion and query times scale linearly with the length of the salt. Notably, the overhead cost of adding salts is minimal, as the Kirsch & Mitzenmacher synthesis allows the salted cryptographic hash to run more quickly than the Murmur algorithm until lengths far longer than the currently-intended 256-bit length of the salt.

VII. CONCLUSION

In this paper, we have explored how analysis of sequencing data can be facilitated by using Bloom filters to implement secure in-the-field pre-processing and data compression. Specifically, we have detailed our proposed SB-FAC architecture for such a system, along with analysis of security concerns, attacker goals, and a range of mitigation strategies, including obfuscation of match values by increasing the false positive rate and protection of Bloom filter contents through salted cryptographic hashing. Mathematical analysis of salted cryptographic hashing indicates that it provides the desired scaling properties. Finally, experimental results show 1) that both cryptographic hash function pairs using either a Kirsch & Mitzenmacher synthesis of hash algorithm sand the Murmur hash algorithm have good uniformity in values, indicating resistance to pre-image attacks and 2) that attackers are expected to incur exponential costs to identify secret salts while the addition of salts adds minimal burden on insertion and query times. Potential future directions for this work include, redteaming to identify potential weaknesses in the architecture and further enhancements to mitigation methods, as well as exploring the range of bioinformatic functions that can be protected in this manner.

REFERENCES

- A. Magi, R. Semeraro, A. Mingrino, B. Giusti, and R. D'aurizio, "Nanopore sequencing data analysis: state of the art, applications and challenges," *Briefings in bioinformatics*, vol. 19, no. 6, pp. 1256–1272, 2018.
- [2] Y. Wang, Y. Zhao, A. Bollas, Y. Wang, and K. F. Au, "Nanopore sequencing technology, bioinformatics and applications," *Nature biotechnology*, vol. 39, no. 11, pp. 1348–1365, 2021.
- [3] A. Kshirsagar, G. Choi, V. Santosh, T. Harvey, R. C. Bernhards, and W. Guan, "Handheld purification-free nucleic acid testing device for point-of-need detection of malaria from whole blood," ACS sensors, vol. 8, no. 2, pp. 673–683, 2023.
- [4] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," ACM Computing Surveys (Csur), vol. 51, no. 4, pp. 1–35, 2018.
- [5] P. Martins, L. Sousa, and A. Mariano, "A survey on fully homomorphic encryption: An engineering perspective," ACM Computing Surveys (CSUR), vol. 50, no. 6, pp. 1–33, 2017.
- [6] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
- [7] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting bloom filters," in *Algorithms–ESA 2006: 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006. Proceedings 14.* Springer, 2006, pp. 684–695.

- [8] M. T. Goodrich and M. Mitzenmacher, "Invertible bloom lookup tables," in 2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton). IEEE, 2011, pp. 792–799.
- [9] F. Bonomi, M. Mitzenmacher, R. Panigrah, S. Singh, and G. Varghese, "Beyond bloom filters: From approximate membership checks to approximate state machines," ACM SIGCOMM computer communication review, vol. 36, no. 4, pp. 315–326, 2006.
- [10] J. Chu, S. Sadeghi, A. Raymond, S. D. Jackman, K. M. Nip, R. Mar, H. Mohamadi, Y. S. Butterfield, A. G. Robertson, and I. Birol, "Biobloom tools: fast, accurate and memory-efficient host species sequence screening using bloom filters," *Bioinformatics*, vol. 30, no. 23, pp. 3402–3404, 2014.
- [11] T. Lemane, P. Medvedev, R. Chikhi, and P. Peterlongo, "Kmtricks: efficient and flexible construction of bloom filters for large sequencing data collections," *Bioinformatics Advances*, vol. 2, no. 1, p. vbac029, 2022.
- [12] E. Seiler, S. Mehringer, M. Darvish, E. Turc, and K. Reinert, "Raptor: A fast and space-efficient pre-filter for querying very large collections of nucleotide sequences," *Iscience*, vol. 24, no. 7, 2021.
- [13] J. Chu, H. Mohamadi, E. Erhan, J. Tse, R. Chiu, S. Yeo, and I. Birol, "Mismatch-tolerant, alignment-free sequence classification using multiple spaced seeds and multiindex bloom filters," *Proceedings of the National Academy of Sciences*, vol. 117, no. 29, pp. 16961–16968, 2020.
- [14] D. Pellow, D. Filippova, and C. Kingsford, "Improving bloom filter performance on sequence data using k-mer bloom filters," *Journal of Computational Biology*, vol. 24, no. 6, pp. 547–557, 2017.
- [15] M. Najam, R. U. Rasool, H. F. Ahmad, U. Ashraf, A. W. Malik et al., "Pattern matching for dna sequencing data using multiple bloom filters," *BioMed research international*, vol. 2019, 2019.
- [16] S. El-Metwally, M. Zakaria, and T. Hamza, "Lightassembler: fast and memory-efficient assembly algorithm for high-throughput sequencing reads," *Bioinformatics*, vol. 32, no. 21, pp. 3215–3223, 2016.
- [17] E. Kucuk, J. Chu, B. P. Vandervalk, S. A. Hammond, R. L. Warren, and I. Birol, "Kollector: transcript-informed, targeted de novo assembly of gene loci," *Bioinformatics*, vol. 33, no. 12, pp. 1782–1788, 2017.
- [18] S. D. Jackman, B. P. Vandervalk, H. Mohamadi, J. Chu, S. Yeo, S. A. Hammond, G. Jahesh, H. Khan, L. Coombe, R. L. Warren *et al.*, "Abyss 2.0: resource-efficient assembly of large genomes using a bloom filter," *Genome research*, vol. 27, no. 5, pp. 768–777, 2017.
- [19] H. Stranneheim, M. Käller, T. Allander, B. Andersson, L. Arvestad, and J. Lundeberg, "Classification of dna sequences using bloom filters," *Bioinformatics*, vol. 26, no. 13, pp. 1595–1600, 2010.
- [20] Z. Zhang and W. Wang, "Rna-skim: a rapid method for rna-seq quantification at transcript level," *Bioinformatics*, vol. 30, no. 12, pp. i283–i292, 2014.
- [21] P. Pérez-Rubio, C. Lottaz, and J. C. Engelmann, "Fastqpuri: highperformance preprocessing of rna-seq data," *BMC bioinformatics*, vol. 20, no. 1, pp. 1–11, 2019.
- [22] D. Wyschogrod, J. Manthey, T. Mitchell, S. Murphy, A. Clore, and J. Beal, "Adapting malware detection to dna screening," in *International Workshop on BioDesign Automation (IWBDA)*, 2022.
- [23] J. Beal, D. Wyschogrod, T. Mitchell, S. Katz, J. Manthey, and A. Clore, "Development and transition of FAST-NA screening technology," Raytheon BBN Technologies, Tech. Rep. BBN Tech Report 8622, 2021.
- [24] Oxford Nanopore, "MinION documentation," https://nanoporetech.com/products/minion, Retrieved Sept 5, 2022.
- [25] K. Christensen, A. Roginsky, and M. Jimeno, "A new analysis of the false positive rate of a bloom filter," *Information Processing Letters*, vol. 110, no. 21, pp. 944–949, 2010.
- [26] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: Building a better bloom filter," in *Algorithms–ESA 2006: 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006. Proceedings 14.* Springer, 2006, pp. 456–467.
- [27] ——, "Less hashing, same performance: Building a better bloom filter," *Random Structures & Algorithms*, vol. 33, no. 2, pp. 187–218, 2008.
- [28] D. R. Stinson, "Some observations on the theory of cryptographic hash functions," *Designs, Codes and Cryptography*, vol. 38, pp. 259–277, 2006.
- [29] B. Donnet, B. Baynat, and T. Friedman, "Retouched bloom filters: allowing networked applications to trade off selected false positives against false negatives," in *Proceedings of the 2006 ACM CoNEXT* conference, 2006, pp. 1–12.