Yale University
Department of Computer Science

Privacy-Preserving Discovery of Consensus Signatures

Felipe Saint-Jean Jian Zhang Joan Feigenbaum
Phillip Porras

YALEU/DCS/TR-1429
July 2010

Privacy-Preserving Discovery of Consensus Signatures

Felipe Saint-Jean* Jian Zhang† Joan Feigenbaum‡
Phillip Porras§

Abstract

Network intrusion detection systems (NIDSs) play a key role in defending modern network computing environments against attacks. NIDS are most typically deployed as perimeter traffic monitors, evaluating packet flows and content against some form of *known* intrusion heuristics, often referred to as intrusion signatures. One key weaknesses with NIDS is their inability to react to "new threats" (i.e. attacks that incorporate content or strategies that are not represented in the current signature set). Unfortunately, the process of generating new signatures is human intensive, thus delaying the reaction time of NIDS to address these new threats. Schemes to generate signatures in an automated way have been proposed with some success, yet their limited accuracy has made them unsuitable for general deployment. In this context, we consider the problem of having a group of semi-trusted parties' share their automatically generated signatures in a manner that preserves each party's privacy, with the goal of rapidly deriving high-quality group consensus signatures that allow the participants to react automatically to an emerging large-scale threat. We first introduce *consensus privacy* as a relevant privacy notion for this setting. We then propose a protocol, based on threshold decryption, to efficiently achieve the consensus privacy notion under realistic assumptions. The output of the protocol is a decrypted subset of those signatures that have been submitted by at least the threshold number

*Yale University, Computer Science Department, New Haven, CT 06520, felipe.saint-jean@yale.edu

†Louisiana State University, Computer Science Department, Baton Rouge, Louisiana 70803, jz@lsu.edu. This work was done while the second author was at SRI.

‡Yale University, Computer Science Department, New Haven, CT 06520, joan.feigenbaum@yale.edu

§SRI, Computer Science Laboratory, Menlo Park, CA 94025, porras@csl.sri.com

of contributors. Our system is more efficient, practical, and scalable than the best-known cryptographic protocol for computing distributed functions in a secure way—at the cost of employing stronger assumptions on an adversary's prior knowledge.

1 Introduction

Consider the following scenario. A network administrator runs an automated signature generation system on his network traffic, which after some period of operation produces a set of signatures. What can he do with the generated signatures? The idea of using these signatures to actively block traffic seems rather perilous, as these unvetted signatures have a potential to disrupt normal networks operations in unexpected ways. The administrator may iteratively test and edit these signatures until some degree of confidence in their quality emerges, but such a manual process significantly impacts the promised value proposition of the automated signature generation system.

Toward the pursuit of new of techniques to derive global threat intelligence, we propose a way to extract value from automatically generated signatures by sharing them. This will make the signatures that represent widespread phenomena attain a visibility that simply cannot be achieved with local analysis, at a speed that cannot be achieved with manually generated signatures. We present a system where sites generate signatures in an automated way, and then contribute these signatures to run a protocol that will detect signatures that are prevalent globally.

Intrusion signatures are often used in computer and network defense systems to detect and block network attacks such as worms. An intrusion signature consists of characteristics of the attack that uniquely identifies it, e.g., content patterns or event-sequence patterns. Recent research in both live traffic signature extraction and static binary content signature extraction has introduced the potential to realize adaptive defenses that can detect, characterize, and incorporate knowledge of newly emerging intrusion patterns to block malicious traffic or to quarantine malicious binary content.

In this paper, we envision future Internet-scale defense infrastructures that can utilize a large distributed pool of contributors that share their locally derived signatures as a means of recognizing emerging threat patterns. Besides providing a global perspective on network attacks, shared signature publishing can also help to consolidate (or prioritize) signatures, i.e., to construct consensus hotlists of signatures from the independently derived sets of local networks. An often-used method for reducing the false positives is to generate signatures only on network flows pre-picked according to strin-

gent criteria that can almost surely determine that the flows are malicious. Clearly, such criteria cannot cover all possible types of malicious flows. In fact, most current generation schemes produce signatures for a few types of attacks. Under our sharing scheme, locally generated signatures need to win a "global consensus" in order to become the final signatures. Therefore, the local generation schemes can relax the flow selection criteria—cover more types of malicious attacks without worrying about false positives. It is at the global level that the signatures are re-examined and the ones with consensus are selected as the final signatures that have fewer false positives.

There have been quite a few security-log sharing systems, for example, DShield [12] collects firewall logs across the Internet and publishes source address blacklists to regularly inform firewalls of the most prolific sources of attack and scan traffic. Signature sharing is different from these systems. Signatures are often extracted by deep inspection of packet content. Automatically generated signatures may capture sensitive information about the local network. There are more significant privacy concerns in signature sharing than in the sharing of other security logs. It is impractical to assume that a signature generation system can protect against the disclosure of private data. Sharing without proper privacy protection leads to an adoption hurdle, if not a direct violation of typical user privacy policies. On the other hand, privacy protection methods widely used for other security-log sharing schemes, such as data sanitization and obfuscation, cannot be applied to signature publication because distorted signatures lose important intrusion characteristics and have little utility. An Internet-scale signature publishing framework must satisfy some basic requirements:

Privacy Protection: The framework should protect the privacy of the contributors from each other and from public users as well as from the framework itself.

Data Utility: Provided that the privacy is protected, the framework should share exact, not obfuscated, information. Distorted signatures have little utility.

Scalable System: The framework should be able to accommodate a large number of data contributors and should disseminate shared information publicly.

We propose a novel signature-sharing architecture with the goal of satisfying the above requirements. To allow better public information dissemination, our architecture takes the form of a data repository. Local networks generate signatures and share them in the repository. Instead of collecting obfuscated signatures, the sharing framework relies on a publication control scheme for privacy protection. The publication control scheme allows

sharing of exact signatures when the privacy constraints have been satisfied. Otherwise, the data is protected. The repository, the other contributors, and the public users of the repository are all unable to obtain the signature.

2 Related Work

There is a wide variety of research on automatic signature generation [11, 5, 7, 10, 3, 9, 2]. Early work explored signature generation using locally repetitive *contiguous substrings* in monomorphic malware traffic. Such systems include Honeycomb [7] (which extracts signatures by Longest Common Substring using suffix-trees), Earlybird [11] (which uses Rabin fingerprints [4] to isolate the most prevalent 40-byte sequences), and Autograph [5] (which also uses Rabin fingerprints to calculate variable-length byte sequences over the sliding payload window). Although similar, these three approaches distinguish themselves in the heuristics used to classify flows or packets as potentially malicious or innocuous. Other techniques include the work of Akritidis et al. [1], who employ the multiplicity of destinations, the length of content substrings (based on sampled Rabin fingerprints), and the position of their appearance within the flow to distinguish worm payload. Polymorphic-based traffic signature generation schemes have also been explored. The Polygraph system [10] extracts multiple disjoint substrings and invariant payload properties, and Hamsa [9] is a high-speed algorithm for similarly extracting multiset substrings. Alternatively, Kruegel et al. employ Content Flow Graphs (CFGs) [8] to extract prevalent executable-code fragments that appear in packets with diverse sources and destinations, matching the executable code structure as a means of resilience to minor polymorphic byte perturbations.

Some research has explored systems for distributed signature generation. Because sharing is done by mutually trusted parties, these systems do not address the privacy of these parties, which is a core problem in our work. For example, Autograph shares address and port dispersion information to accelerate flow classification among the monitors as they select packets from which to extract repetitive content. WormShield [3] shares address dispersion statistics in a Zipf-like calculation along with content fingerprints, and forwards this information to the root node to manage signature consolidation. Since the sharing is among mutually trusted parties, these systems do not address the privacy concerns in such sharing, which is the core problem of our work.

In the area of privacy-preserving computation there are two protocols related to our work. The first is Yao's Secure Function Evaluation [13].

This is a general protocol for privacy-preserving computation and could be used to share signatures. Yao's protocol is extremely powerful and secure, but does not scale well to our needs. It is too expensive. A more efficient option is the more specific protocol for privacy-preserving set operation [6]. These protocols are among the most efficient for computing set operations in a privacy-preserving way, but their communication complexity is quadratic to the number of inputs. For the volume of signature comparisons that will arise in event modest deployments of our our scheme, $O(n^2)$ may prove too expensive for scalable signature publication.

3 Problem Description

Our primary challenge is to develop a system to share signatures, in a massive way, that can be implemented in practice where the participants get a valuable result for their involvement. We want to avoid imposing a heavy burden on the contributors or to put their private data at risk. But we must also complete our consensus computation regardless of the sudden absence or introduction of contributors into an evaluation round.

This leads to a more specific set of objectives:

Transient contributors: we must allow for a wider range of contributors. We want interactions to be short and have the protocol to be robust in results generation, even when some contributors fail to finish or to show up at all.

Privacy: the privacy of contributor data is fundamental in finding people willing to participate. Network data contains private information, and thus signatures may incorporate this private private data.

Scalability: any solution must facilitate data collection volumes that allow a publication to monitor global phenomena. We want the system to perform with thousands of contributors each submitting hundreds of signatures per round.

Privacy, being in general an elusive concept, has a very defined meaning in our system. We define *consensus privacy* for a value t in the following way. If t or more contributors present the same signature, it is not private and thus should be disclosed exactly as a result of the protocol. This definition is based on the idea that private network data is *unique* to the site where that piece of data is observed. If a signature incorporates private data, that data (being unique to that site) should not be revealed, since the

chances of $t-1$ other sites contributing the same signature are extremely low (coincidental with low thresholds, and increasingly improbable as our threshold increases). The consensus privacy concept precisely defines what should and what should not be published as a result of the protocol.

What the system will compute is the set of signatures that were generated at the sites of many of the contributors. More precisely, we want to compute the *threshold union* set on the contributors' inputs at each round. The threshold union set is the set that has all input elements that are present more than t times in the union of all contributors' inputs. In this case, the inputs are the locally generated signatures. In the more general problem, contributors can input multiple copies of an element; we only consider the case where each contributor submits at most a single copy of each possible element. This function can be described as computing the subset of all signatures that are present in at least t of the contributors' locations. This function arises naturally from the consensus privacy notion introduced previously. The challenge is to achieve this notion of privacy in a distributed system with semi-trusted parties.

Protocols that solve the threshold union problem with very strong privacy properties are known [13, 6]. These general protocols do not adjust to our requirements. They are multi-round protocols and have a high communication complexity. To achieve our goal we can either formulate a new protocol that improves the upper bound on the problem, or we can relax the assumption. Here we explore the latter option, and do so in the following way. We classify signatures into two groups, a signature is either a *true signature*, meaning that it captures that phenomena that has been observed from multiple distributed locations, or it is a *private signature*, a signature that has not yet proven to be applicable outside a single network location. A true signature can be generated, identically, in many sites yielding few or no privacy concerns. A *private signature*, on the other hand, contains information of the local network traffic and becomes a signature by error. It is very specific to the site where it was generated, and thus the privacy implications are sources of greater concern. From this perspective, privacy protection will be focused on limiting the degree to which an adversary may gain insight into the contents of private signatures. We will assume that positives have high minimum-entropy in relation to the adversary's knowledge. That is, the information revealed by the execution of the protocol will not help an adversary with little specific knowledge of the internal traffic.

We propose a simple communication model in which contributors can encrypt/encode and send each signature prior to dissemination to our consensus repository. Thus, the communication complexity for each signatures

encoding is distributed across the contributor pool, and each the computations that each contributor performs are a function of the amount of signatures they choose to disseminate. A centralized party collects all the signatures and can reveal, without perturbation, the value of signatures with a frequency equal to or above the threshold t. In the following section we present a scheme that satisfies the requirements under this communication model.

4 System Description

The objective of the system is to compute, in a distributed fashion, the threshold union function among the private inputs of a set of participants. We will do this in a centralized model because it fits better with the conditions necessary to make the system practical, not only in terms of low complexity, but also in the sense that network administrators will find it compelling to participate as contributors.

The following roles are involved in the protocol:

The Contributors are the nodes that represent each of the networks where data is being collected. These contributors are running an automatic signature generation program on their network traffic. Their inputs to the protocol are the signatures generated by it. Let us assume that there are n contributors and they are called $\mathcal{S}_1, ..., \mathcal{S}_n$.

The Collector is the centralized party responsible for receiving the inputs, computing the threshold union, and subsequently publishing the results in a way that the contributors can access them. We will call the collector $\mathcal{C}$.

The Registration Authority is responsible for generating a distributing public and private keys to the contributors. We call the registration authority $\mathcal{RA}$.

We envision the protocol running once per fixed period of time. It could run, for example, once a day. During that day the contributors collect and submit to the collector signatures in an encrypted form. At the end of each collection period, $\mathcal{C}$ runs the decryption process to reveal only those signatures satisfying the consensus privacy condition.

The contributors interact with $\mathcal{RA}$ once at the very beginning. From then on, the contributors interact only with the collector, to submit signatures and to get results. Figure 1 summarizes the interaction between the parties involved.

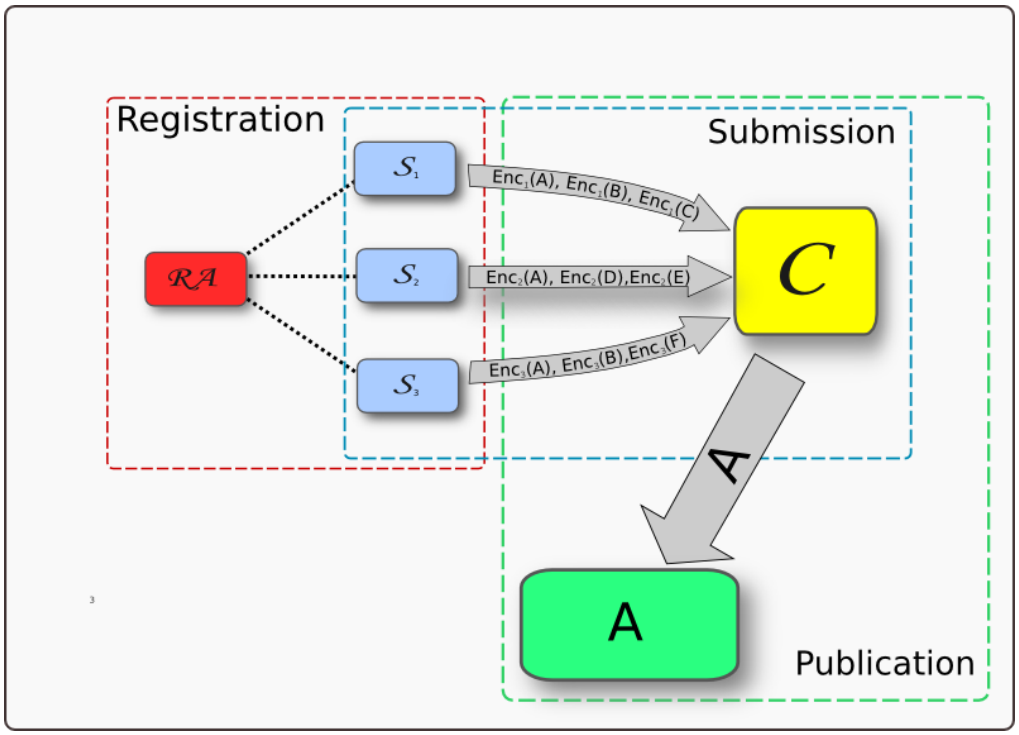

Figure 1: General architecture description: $\mathcal{RA}$ and $\mathcal{S}_i$ interact only once in the registration stage. During the submission stage, the contributors $\mathcal{S}_1, \mathcal{S}_2$ and $\mathcal{S}_3$ send the encryptions of their collected signatures to $\mathcal{C}$. In the last stage, $\mathcal{C}$ decrypts and publishes the signatures that have consensus. In the diagram, only signture A is published for consensus threshold $t = 3$.

We now examine two potential protocols to achieve consensus privacy under the sharing scheme described above, and consider the second protocol to be the preferred strategy.

4.1 Version One

Given our consensus privacy notion, it seems natural to apply some of the threshold cryptography theory in order to achieve a system that respects our consensus-based privacy definition. A threshold cryptosystem allows for the private key to be shared among n parties, and any set of size t can decrypt a ciphertext. We assume the existence of the following (t, n)-threshold decryption scheme:

- $KeyGen(1^k)$: The key generation algorithm that generates the private key shares $\{\mathsf{Pri}_i\}, i = 1...n$ and a public key Pub.
- $Encrypt(a, \mathsf{Pub})$: Given a message a and a public key Pub it generates the ciphertext C.
- $PartialDecryp(\mathsf{C}, \mathsf{Pri}_i)$: Given a ciphertext C and a share of a private key Pri_i it computes C_i, a partial decryption of C.
- $Combine(\{\mathsf{C}_i\}_\mathsf{T})$ with $|\mathsf{T}| = t$: Given partial decryptions $\{\mathsf{C}_i\}_\mathsf{T}$ it reveals the plaintext message a.

To use a (k, n)-threshold scheme, like the one above, to share signatures we can do the following. First, $\mathcal{RA}$ generates the keys and distributes them to the participants, giving Pri_i to $\mathcal{S}_i$. To submit a signature a, node $\mathcal{S}_i$ will encrypt it with the public key and then compute the partial decryption of it as $\mathsf{C}_i = PartialDecryp(Encrypt(a, \mathsf{Pub}), \mathsf{Pri}_i)$, using its share of the private key Pri_i. When $\mathcal{C}$ combines t partial decryptions belonging to the same signature, $\mathcal{C}$ can then reveal a, the original signatures, by computing $Combine$ on them. This defines a new function

- $Share(a, \mathsf{Pri}_i) = PartialDecryp(Encrypt(a, \mathsf{Pub}), \mathsf{Pri}_i)$. It generates a share of a that will allow reconstruction using $Combine(.)$. This is, in practice, a reusable secret-sharing scheme.

This system is described by the following functions:

Submission: A node $\mathcal{S}_i$ on a signature a will submit $\beta = Share(a, \mathsf{Pri}_i)$.

Revelation: $\mathcal{C}$, in order to publish, needs to decrypt. For that, it one needs to find t submissions that belong to the same signature and then $Combine(.)$ them.

$\mathcal{C}$ will be able to decrypt valid signatures that show up more than t times, because in order to get a correct decryption, $\mathcal{C}$ needs to group together t encryptions of the same signature. It is not possible to do that on ciphertext, since that would imply that the ciphertext reveals information about the plaintext. In the next section we present a variation that improves efficiency by revealing some information about the signature in a controlled way.

4.2 Version Two

One strategy to improve the decryption step is to attach to partial decryptions some information that will allow $\mathcal{C}$ to group signatures effectively. Attaching a short k-bit hash $\mathsf{H}_l(a)$ to the partial decryption will allow $\mathcal{C}$ to group candidate signatures together that have the potential to equal. Conceptually, one can think of $\mathsf{H}_l(a)$ as the bin to which the signature a is assigned. The value of k has to be tuned to give enough information to group together a set of encryptions of the same signature that are very likely to correspond to the same signature, but not enough information to reveal a signature that is truly a unique locally private signature. This can be done effectively given the observation that the empirical distribution of signatures reveals a Zipf-like distribution on signatures received by $\mathcal{C}$. That is, we find that there are many elements of a signature that are highly repetitive across a corpus of signatures, and those will distribute across the bins. A few things will repeat a lot, and those will dominate the bin they are in, allowing decryption by sampling t elements at random and attempting a decryption. This observation is based on preliminary experiments and requires further grounding; a less convenient distribution, like a uniform one, means that we need more bits in H_l, but the scheme will still work.

Submission : A node $\mathcal{S}_i$ on a signature a will submit the pair $Enc_i(a) = (\alpha = \mathsf{H}_l(a), \beta = Share(a, \mathsf{Pri}_i))$.

Revelation : $\mathcal{C}$ will group all submissions with matching α and will decrypt by sampling. When a signature is correctly decrypted, then it is published. The verification of decryption correctness can be done by adding some redundancy to a.

5 Efficiency

The step that may affect the scalability of our system is the identification of consensus. In this step, because we use a hash with a small number of bins H_l, there may be collisions, which may increase the effort we need for recovering the consensus signatures. We now analyze the relationship between the number of bins, $\mathcal{B}$ (note that $k = \log \mathcal{B}$), used in the hash and the amount of effort we need to decrypt the consensus signatures. In the collection of signatures, we have both consensus signatures (we denote the number of distinct consensus signatures by $\mathcal{T}$) and non-consensus signatures (let $\mathcal{N}$ be the number of distinct non-consensus signatures). Because our sharing process cares about only consensus signatures, all the non-consensus signatures can be viewed as noise signatures. In our sharing scenario, $\mathcal{N} >> \mathcal{T}$ we use $\mathcal{B} > \mathcal{T}^2$ number of hash bins such that with probability larger than $\frac{1}{2}$, there is no collision, i.e., each consensus signature is placed into a separate bin. In practice, the consensus threshold t often takes a small value (say below 10). Therefore, we treat the threshold as a small constant in the analysis.

If we use $\mathcal{B} = \mathcal{N}$ number of hash bins, the number of noise signatures in a bin follows a Poisson distribution. The expected number of noise signatures is 1. With probability larger than $\frac{1}{2}$, we can bound the number of noise signatures in any bin to be smaller than a small constant $c > 1$. Now consider a bin that contains a consensus signature. We have at least t copies of that consensus signature. There are also at most c distinct noise signatures in this bin, and each noise signature has less than t copies. The signal-to-noise ratio is above $k/(kc) = 1/c$, and the fraction of consensus signatures in the bin is at least $1/(1 + c)$. If we randomly select t shares from the bin, the probability that they come from the same private field a is $[1/(1 + c)]k$. Therefore, we need to make $(1 + c)k$ such selections to decrypt and recover the consensus private field a.

We note that the above analysis is about the worst-case decryption effort. The actual effort required in practice can be much smaller. Figure 2 plots a simulation result about the relationship between the bin number and the decryption effort. In the simulation, 1000 signatures were generated using a Zipf distribution of parameter 1000, i.e., the frequency of the i-th signature (the number of contributors that generate that signature) is $1000/i$. We chose a Zipf distribution because the frequencies of the signatures produced in our preliminary experiments on our local network's traffic follow this distribution. However, we note that this is an example simulation and that the scalability of our system does not depend on a particular signature

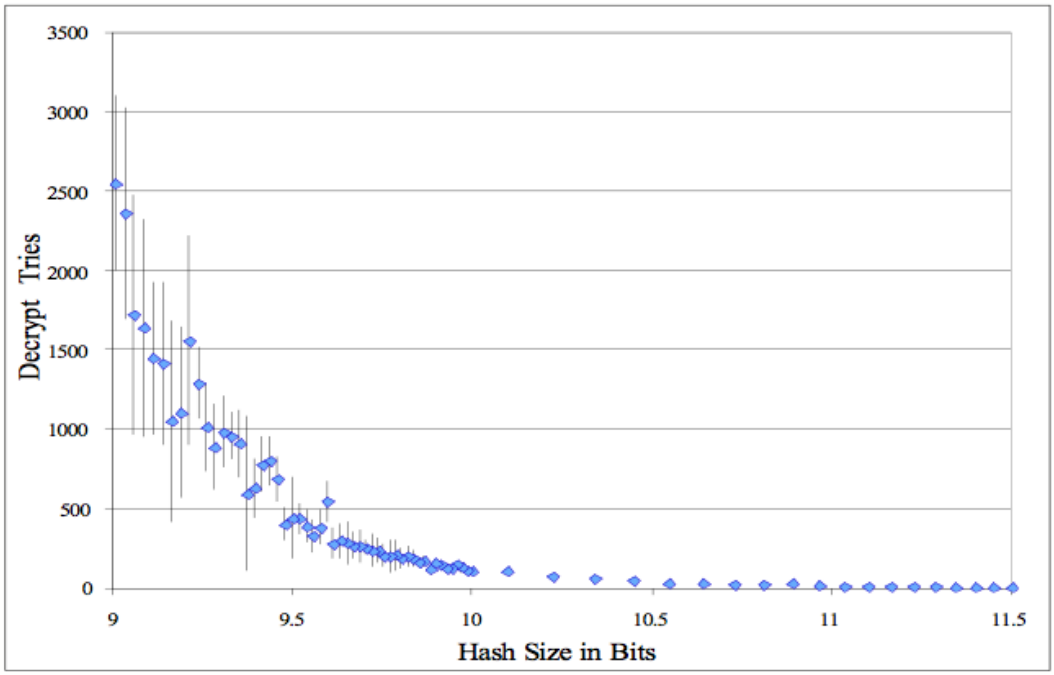

Figure 2: Decryption effort for a number of bits in the hash

distribution. Once generated, each signature is hashed into a random bin among $2k$ bins where k is the size of the hash in bits. We then locate each signature whose frequency is above $t = 5$, and compute the number of tries we need to decrypt that signature given the condition of the bin. Figure 2 plots the number of tries versus t. For each t, the simulation is conducted 20 times, rehashing the signatures each time. The average effort (blue diamonds) and the standard deviation (vertical bars) are plotted. We see that when the number of bins used in hashing (say $2^{10} = 1024$) is close to the number of unique signatures (1000), we can successfully decrypt a consensus signature after only a small number of tries.

Both the analysis and the simulation result show that when using a number of hash bins that is in the same order of the unique signatures, the effort required for decryption is well within the capability of a modern computer. In practice, we expect the decryption effort to be even smaller, because only the private fields in a signature are converted into the protected form and thus require the search for consensus. Signatures normally contain other fields that are not privacy sensitive. Information from these fields can

be used to further speed the search for consensus. For example, if one signature is about TCP traffic and the other is about UDP, the repository does not need to test consensus for the private fields between these two signatures.

6 Security

We define our adversary $\mathcal{A}$ as a polynomially bounded adversary that has access to all the information $\mathcal{C}$ has, but limited knowledge about the details of the internal traffic of any individual contributor. We define a security breach as the adversary gaining too much knowledge about a non-consensus signature, i.e., a signature submitted by fewer than t contributors. By assuming strong enough properties of the threshold cryptosystem used (ind-cca, which means the ciphertexts are indistinguishable in a chosen ciphertext attack), and having H_l be a random oracle, we can provide the following security argument.

We will challenge the adversary to guess the plain signature. We will say that the adversary fails if he can guess correctly with a small probability, smaller than $\frac{1}{2^s}$ for a security parameter s. For example, consider the following scenario. The adversary gets a set of partial decryptions of a signature a, and then makes a guess m as to a candidate for a. Because H_l is a short hash, he has no trouble finding a collision, so $\mathsf{H}_l(m) = \mathsf{H}_l(a)$. How good is his guess after seeing the hash? This is defined by the following conditional probability:

$$\begin{aligned}\mathbb{P}[a = m | \mathsf{H}_l(a) = \mathsf{H}_l(m)] &= \frac{\mathbb{P}[a = m, \mathsf{H}_l(a) = \mathsf{H}_l(m)]}{\mathbb{P}[\mathsf{H}_l(a) = \mathsf{H}_l(m)]} \\ &= \frac{\mathbb{P}[a = m]}{\mathbb{P}[\mathsf{H}_l(a) = \mathsf{H}_l(m)]}\end{aligned}$$

We assume that the adversary's knowledge regarding the possible signatures has min-entropy h, thus

$$\mathbb{P}[a = m] \leq \frac{1}{2^h}$$

and, because the H_l values are assigned uniformly at random (random oracle property),

$$\mathbb{P}[\mathsf{H}_l(a) = \mathsf{H}_l(m)] = \frac{1}{2^k}$$

so,

$$\mathbb{P}[a = m | \mathsf{H}_l(a) = \mathsf{H}_l(m)] \leq \frac{1}{2^{h-k}}.$$

If we have that $h - k \leq s$, then the adversary can only guess correctly with a negligible probability. This formalizes the intuition that if we have h bits of min-entropy, and we reveal k bits, we are left with $h - k$ bits of min-entropy. In that way, $h - s$ is our hash-bit budget for performing an efficient decryption.

The previous argument holds when the adversary is challenged to guess the exact signature. But what about a partial guess? For example, what about predicates. Because the cryptosystem used is $ind - cca$, all signatures with the same value of H_l are indistinguishable. Furthermore, because hash values are assumed to be assigned at random, any sample of signatures the adversary uses to guess a predicate will be roughly evenly distributed, making his guess very close to random. So under this assumptions the adversary cannot guess even single bits.

We now discuss some potential adversarial actions against the sharing system. A malicious contributor may attempt to recover protected signatures by submitting arbitrary data. However, this is impossible because consensus is required to reveal the signatures in the system. In other words, a malicious contributor can submit arbitrary signatures. This may make the discovery of consensus signatures very difficult for the repository but it cannot break the privacy of the other contributors. To recover the protected signatures, the malicious contributor has to launch a dictionary attack. In this attack, the malicious contributor needs to submit many possible signatures. The contributor has to collude with $t - 1$ other contributors, each of them also submitting the same set of possible signatures, to achieve consensus. These conditions, although they cannot completely prevent dictionary attack, make it extremely difficult and easy to be detected. We also note that the protected form of the private fields is time dependent. Consensus can be achieved only if the signatures are supplied within the same time window. Hence, a dictionary attack has to be launched for each time window. This also greatly increases the work factor of such an attack.

7 Conclusions and Further Work

We presented the design of a system to share network signatures that protect contributor privacy in a realistic and practical way. We envision implementing this system (or an enhanced version of it), and with it achieve two major goals. First, it would provide signatures of global attacks fast and with low cost. Because of the privacy and efficiency properties we expect a large number of contributors, so the produced signature set provides a valuable

picture of new attacks. Also, having a system like this in place allows for security research to move in directions that, so far, have been restricted because of the lack of data. Data has been hard to obtain because of privacy concerns. If our system is successful in recruiting contributors, the generated signature dataset will be of extreme value to security research in gaining an understanding of global phenomena on real active networks.

References

[1] P. Akritidis, E. P. Markatos, M. Polychronakis, and K. G. Anagnostakis. STRIDE: Polymorphic sled detection through instruction sequence analysis. In *20th International Conference on Information Security (SEC)*, pages 375–392, 2005.

[2] David Brumley, James Newsome, Dawn Xiaodong Song, Hao Wang, and Somesh Jha. Towards automatic generation of vulnerability-based signatures. In *IEEE Symposium on Security and Privacy*, pages 2–16, 2006.

[3] Min Cai, Kai Hwang, Jianping Pan, and Christos Papadopoulos. Wormshield: Collaborative worm signature detection using distributed aggregation trees. Technical report, Internet and Grid Computing Lab, Univ. of Southern California, 2005. `http://gridsec.usc.edu/TR/TR-2005-10.pdf`.

[4] B. Karp and M.O. Rabin. Efficient randomized pattern-matching algorithms. *IBMJRD: IBM Journal of Research and Development*, 31:249–260, 1987.

[5] Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *USENIX Security Symposium*, pages 271–286, 2004.

[6] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *Advances in Cryptology (CRYPTO)*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257, 2005.

[7] Christian Kreibich and Jon Crowcroft. Honeycomb: Creating intrusion detection signatures using honeypots. *Computer Communication Review*, 34(1):51–56, 2004.

[8] Christopher Krügel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. Polymorphic worm detection using structural information of executables. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 3858 of *Lecture Notes in Computer Science*, pages 207–226, 2005.

[9] Zhichun Li, Manan Sanghi, Yan Chen, Ming-Yang Kao, and Brian Chavez. Hamsa: Fast signature generation for zero-day polymorphic-worms with provable attack resilience. In *IEEE Symposium on Security and Privacy*, pages 32–47, 2006.

[10] James Newsome, Brad Karp, and Dawn Xiaodong Song. Polygraph: Automatically generating signatures for polymorphic worms. In *IEEE Symposium on Security and Privacy*, pages 226–241, 2005.

[11] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated worm fingerprinting. In *Proceedings of the 6th Symposium on Operating System Design and Implementation*, pages 45–60, 2004.

[12] Johannes Ullrich. Dshield home page. `http://www.dshield.org`, 2008.

[13] Andrew Yao. Protocols for secure computation. In *Proceedings of 23rd Annual Symposium on the Foundations of Computer Science*, pages 160–164, 1982.