Poster Session Proceedings

8th IEEE European Symposium on Security and Privacy

Preface

This volume contains the abstracts of the posters accepted and presented at the EuroS&P 2023 conference, which was held on July 3-6, 2023 in Delft, the Netherlands. Following the tradition initiated with the 2022 edition of the conference, these proceedings are published on Zenodo (https://zenodo.org), an open research repository operated by CERN, which enables sharing and preserving of research outputs.

The poster session of Euro S&P is conceived as an opportunity for security and privacy researchers to share their recent results, and to obtain valuable feedback on their ongoing work from participants at the conference. In total, 8 abstracts were submitted to the poster session, which underwent a lightweight review process. In particular, reviews were aimed at checking the coherence of the abstracts with the scope of Euro S&P, rather than providing in-depth technical feedback as for a regular submission. Each abstract was reviewed by at least two members of the program committee. After the review phase, the program committee decided to accept all submitted abstracts.

I would like to thank all the authors for their submissions, and especially the poster presenters to have brought up to life an interesting poster session at the conference. I am also indebted to the program committee of the Euro S&P poster session, for their timely and useful reviews. I would also like to thank the general chairs of Euro S&P 2023, Kaitai Liang and Georgios Smaragdakis; the program chairs David Evans and Herbert Bos; the publication chairs Luca Verderame and Stefanie Roos. Finally, I am also grateful to Ivan Liang for setting up the HotCRP platform for managing the submission process, and to the web chair Roland Kromes for updating the website with the updates related to the call for posters.

Luca Mariot, University of Twente, the Netherlands
IEEE European Symposium on Security and Privacy 2022 Poster Chair

Program Committee

Poster Chair

Luca Mariot, University of Twente, the Netherlands

PC Members

Michele Carminati, Politecnico di Milano, Italy
Andrea Continella, University of Twente, the Netherlands
Alessandro Erba, CISPA Helmoltz Center for Information Security, Germany
Claudio Ferretti, University of Milano-Bicocca, Italy
Roland Kromes, Delft University of Technology, the Netherlands
Azqa Nadeem, Delft University of Technology, the Netherlands
Mario Polino, Politecnico di Milano, Italy
Martina Saletta, University of Trieste, Italy
Fatih Turkmen, University of Groningen, the Netherlands

List of Accepted Posters

1. Privacy-Preserving Heat Map Generation through Spatial and Temporal Local Perturbation
   Toon Dehane, Michiel Willocx, Bert Lagaisse, Vincent Naessens

2. Towards a Secure and Practical System to Obfuscate Tor Network Traffic
   Minjae Seo, Myoungsung You, Taejune Park, Seungwon Shin, Jinwoo Kim

3. Using Causal Inference to Extract Hidden Information from Dependency Modelling
   Ayodeji Rotibi, Neetesh Saxena, Pete Burnap

4. Don’t Trust, Verify: Empowering Last-Mile Security and Privacy in Web3
   Weihong Wang, Tom Van Cutsem

5. Are we reasoning about cloud application vulnerabilities in the right way?
   Stefano Simonetto, Peter Bosch

6. Tight Short-Lived Signatures
   Arup Mondal, Ruthu Hulikal Rooparaghunath, Debayan Gupta

7. RANDGENER: Distributed Randomness Beacon from Verifiable Delay Function
   Arup Mondal, Debayan Gupta

8. MaSTer: (Practically) Maliciously Secure Truncation for Replicate Secret Sharing
   Martin Zbudila, Erik Pohle, Aysajan Abidin, Bart Preneel

Poster: Privacy-Preserving Heat Map Generation through Spatial and Temporal Local Perturbation

Toon Dehaene, Michiel Willocx, Bert Lagaisse, Vincent Naessens
DistriNet, KU Leuven
Belgium
firstname.lastname@kuleuven.be

***Abstract*—Businesses and governments increasingly apply machine learning and AI techniques for various strategic optimization purposes. Many applications, however, require location data over time of many individuals to train models. Currently, location streams are collected – often without the end-user's consent or even knowledge – by means of seemingly innocent smartphone applications that continuously report location data in a background process, raising serious privacy concerns. In this poster, we propose a transparent and privacy-friendly approach that perturbs sensitive location data in two dimensions, namely spatially and temporally. The mechanism can be deployed on mobile devices such that all obfuscation is executed locally, thereby protecting the user even in the presence of an untrustworthy data collector. The viability of the proposed mechanism is demonstrated by generating heat maps and predicting hot spots based on privacy-preserving location streams of many users.**

***Index Terms*—Location, Privacy, Obfuscation, Perturbation**

I. Introduction

Nowadays, many companies and governmental institutions rely on AI and ML models to optimize business goals and societal challenges. These models offer, amongst others, insights in busy areas and crowd movements and aim at optimizing emerging challenges based on the findings. Examples are traffic optimization based on historic data streams, optimizing visitor flows in touristic hot spots, selecting feasible locations to open new shops, bars or restaurants and deliberately assigning first aid personnel during events.

Currently, location streams are collected and aggregated by data brokers, and subsequently sold to data analysts. Data collection often occurs by means of smartphone applications. Dedicated data collection processes typically run in the background, continuously logging user locations, and subsequently send the data to a remote server. Users are often completely unaware of these practises. Although privacy legislation and technology providers are cracking down on these practices, industry practises show that this type of data is still sold on a massive scale today at various platforms.

While collecting personal location data over time is controversial and impedes the user privacy, the practical applications of location data are often of a legitimate nature and provide immeasurable value to governments and commercial organizations. To improve the privacy/utility balance, this poster proposes a privacy-friendly alternative for collecting location data over time. The mechanism demonstrates that – while a feasible privacy level is reached – crowd insights can still be learned from the obfuscated data. The mechanism performs all obfuscation locally on the user's device, thereby distrusting data collectors.

Contributions. This poster proposes a hybrid mechanism that supports the privacy-preserving collection of location streams. We introduce the concept of automatically generated privacy blobs, which vastly limit the amount of information an individual leaks towards data collectors, even when reporting location data over time. While the location data of each individual separately is meaningless up to a certain privacy level, we demonstrate that aggregating the privacy-enhanced location data still allows the creation of accurate heat maps on a population-level.

II. Related work

Many research already focused on location obfuscation. Early work [4], [6] was mostly concerned with obfuscating location datasets of known size. Later works [8] have raised concerns about the continuous release of locations, and provide privacy guarantees for location streams. Our approach benefits from these research results.

For location obfuscation, our work builds further upon geo-indistinguishability research that explores noise addition [1], and adopts the concept of Privacy Zones used by fitness tracking social networks such as Strava. Several vulnerabilities have been found in existing implementations of Privacy Zones [5], [7]. This work fine-tunes Privacy Zones in accordance with the proposed countermeasures.

Our research provides a privacy-friendly alternative for privacy-invasive data analysis on accurate location data, without losing substantial prediction accuracy of crowd optimization purposes. Therefore, privacy and utility [3], [9] need to be balanced. Lastly, our solution is highly configurable, providing users with transparent control over their privacy settings and providing developers with the necessary tools to cover a wide variety of use cases [2].

III. Location obfuscation mechanism

A privacy blob is the union of multiple privacy zones, which defines a finite set of historically reported zones on the map. To support reporting over time, the existing set of privacy zones is only extended if a new location outside the existing zones is visited. Otherwise, the reported location will be at the midpoint – and optionally radius – of a privacy zone already present in the privacy blob. Furthermore, we put constraints on the reporting frequency. The *major design goals* are enumerated below. A first one is to support continuous reporting of location data without unacceptable privacy degradation. The attacker model assumes a distrusted data collector. To achieve this goal, all obfuscation occurs locally, which is performed by constraining reports to a limited set of location points. A second goal is offering the potential for creating meaningful aggregated heat maps. For this purpose, each user keeps a unique yet limited set of coordinates to report from. Averaging them for a crowd allows to approximate density calculation thereby relying on data aggregation. Privacy settings – like radius and report frequency – can depend on the location sensitivity and time. The user sets a default privacy zone radius and reporting frequency for automatically generated zones. A third design goal is to support customization for sensitive locations and time intervals, and combinations thereof. For instance, a user can opt not to report her location between 1 a.m. and 8 a.m. We support user-created privacy zones with customized radius and frequency, that further do not differ conceptually from the automatically generated ones. For example, the default radius can be 200 meters, but can be extended to 500 meters near her house. Multiple privacy zones of a user that are part of the privacy blob can have overlapping areas. If all privacy zones that make up the intersection would be released, a much smaller area of possible real locations should be leaked to the attacker. Similarly, we do not want to report the set of privacy zones successively in a short time span, as this would leak with high probability that the user is in the intersection area. The obfuscation mechanism contains logic to introduce low-pass in the choice of her reported privacy zone. This has the effect that the user has a high chance of only reporting a single location even when in the intersection of multiple privacy zones, therefore not leaking that she is in the intersecting area. The *flow of the proposed strategy* is as follows. First, a time check asserts a minimum interval between two subsequent reports. It also checks against any user-set time rules. Secondly, the set of all privacy zones from which a report has been made in the past is evaluated. These are $(location, radius)$ tuples covering the map. The zones that contain the current real location are filtered.

If that set is empty, a new privacy zone with the default radius that contains current location is created, and subsequently reported. If the filtered set is not empty, a random one is selected and its midpoint is reported. When picking from multiple privacy zones, the following strategy is applied. If the previous reported zone is in the set, there is a high probability that it is picked again. Optionally, additional perturbation is done in the time domain by adjusting the reported timestamp.

IV. Creating heat maps – preliminary experiments

This section demonstrates that location data, obfuscated with the privacy mechanism described in the previous section, still allows to create meaningful heat maps for crowd insights. The experimental setup is presented in Figure 1 and initially abstracts the time dimension. Firstly, a synthetic dataset of locations is created, in which one or multiple ongoing events are simulated by increasing the density of points in certain areas of the map. Next, the proposed privacy mechanism is applied to the dataset. A heat map is constructed of both the original and the obfuscated dataset. Our experiments demonstrate that it is feasible to reconstruct the simulated events reflected by hot spots on a map with a high level of accuracy after applying the proposed transformation to the data.

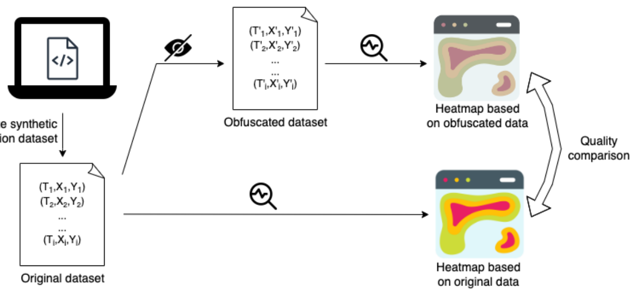


Fig. 1. Experimental setup for creating and comparing heat maps based on original and obfuscated data.

The heat maps are created as follows. For the current simulation, we assume a population that is uniformly distributed over a square map with a density that is representative for many cities. We then generate points for an event of N people normally distributed with a predefined average distance from a common center. We add the people attending the event to the background density of the city. We then apply the obfuscation mechanism and get a perturbed population. We generate a heat map for the original and perturbed population and compare them in a final step. The settings used for figures 2 and 3 are as follows. Event A has 1000 visitors with an average radius from the center of 200 meters. Event B has 200 visitors with an average radius from the center of 100 meters. The background population is 5457 people per square kilometer. In figure 2 people belonging to an event are moved between 20 and 80 meters to the right and up every iteration, 50 times in total. The background is a snapshot at iteration 24. All users generate privacy zones with a radius of 300 meters, and do not have a maximum report frequency.

A second experiment envisions slow movements of individuals attending the event. This leads to traces for each individual. Each user acts according to the mechanism described earlier, independent from others. The original as well as the perturbed traces of a single user are depicted.

Finally, the quality of the heat maps is assessed. With respect to utility, preliminary results show that heat maps can be reconstructed up to a representative degree. This is supported by comparing the density plots generated from the original data (left in figures 2 and 3) to the ones generated from the perturbed data (right in figures 2 and 3). The density reveals the magnitude of larger-scale events only, while the exact amount of people attending an event is not exposed. More quantitative testing is necessary.

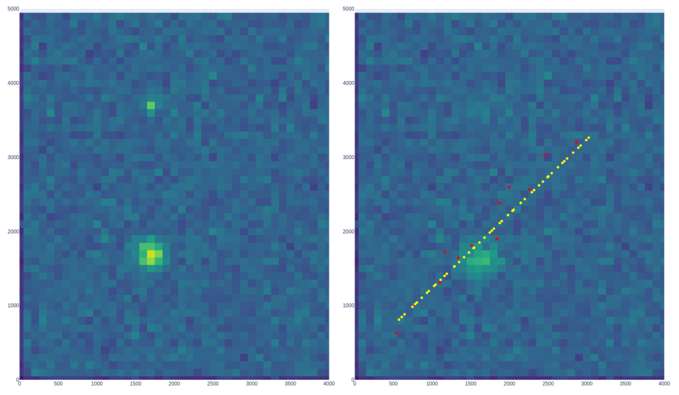


Fig. 2. Left: density of the real data. Right: density of the perturbed data. Yellow markers: real data. Red markers: perturbed data. (X-axis and y-axis in meters. Color axis is density per area, lighter is more dense)

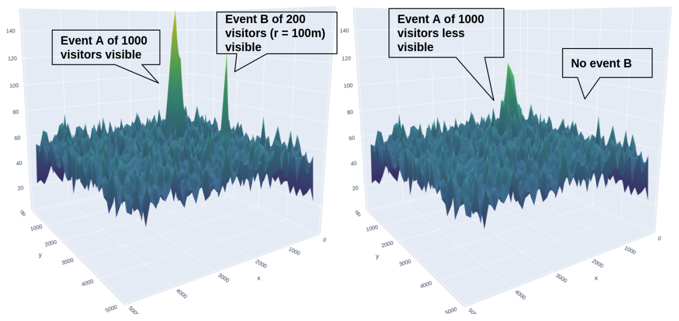


Fig. 3. Left: density of the real data. Right: density of the perturbed data. (X-axis and y-axis in meters. Z-axis density per area).

Regarding privacy, preliminary results show that it is hard to predict the real location history based on the perturbed data streams. This claim is supported by checking the distribution of the errors between reported and real locations (figure 4).

V. Conclusion and Future Work

This poster presents the preliminary experimental results on privacy preserving crowd monitoring. Our privacy-preserving mechanism dynamically creates privacy blobs. Each original location is mapped to the midpoint of a privacy zone in the blob. Whenever a user moves within a previously created privacy blob, the corresponding midpoint is always reported to the server. This mechanism ensures that individuals never unintentionally leak their exact location. While this method aims at protecting the user's location privacy over time, our research also demonstrates that aggregate data – over a large set of users – still contains an acceptable utility level for crowd analytics purposes.

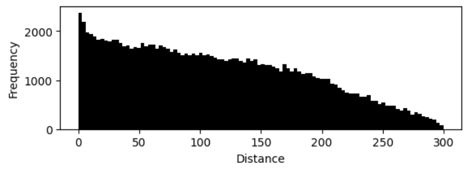


Fig. 4. Distribution of distances between real and perturbed data points

Our approach is validated by creating heat maps from obfuscated location data of a set of artificial users by our privacy mechanism, and compares these heat maps to heat maps of the original datasets (i.e. before obfuscation). Currently, The heat maps included in this poster confirm a basic appropriate level of accuracy after applying obfuscation. Formal metrics are required to thoroughly assess our initial findings.

Realistic case studies and data sets can further support our claims. This will be done in the retail sector in which an industry partner predicts feasible locations to open new stores. Accuracy will be compared between predictions made on the real and the perturbed data.

Another research direction consists of developing an offensive model that aims at recovering as much detail as possible from the obfuscated dataset. This model should have complete knowledge of the perturbing mechanism, and have access to test sets of real location data in the training stage. The output of this offensive mechanism when applied to obfuscated data can then be combined with the original data to produce comparative metrics. The obfuscating mechanism can then be more easily tuned by iterating over different parameter values, finding pareto improvement in the utility-privacy plane.

This research is partially funded by the VLAIO ICON project CoCoNUT and the SOLIDLab Flanders initiative.

References

[1] Miguel E Andrés, Nicolás E Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Geo-indistinguishability: Differential privacy for location-based systems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 901–914, 2013.

[2] Sophie Cerf, Vincent Primault, Antoine Boutet, Sonia Ben Mokhtar, Robert Birke, Sara Bouchenak, Lydia Y Chen, Nicolas Marchand, and Bogdan Robu. Pulp: Achieving privacy and utility trade-off in user mobility data. In *2017 IEEE 36th symposium on reliable distributed systems (SRDS)*, pages 164–173. IEEE, 2017.

[3] Konstantinos Chatzikokolakis, Ehab Elsalamouny, and Catuscia Palamidessi. Efficient utility improvement for location privacy. *Proceedings on Privacy Enhancing Technologies*, 2017(4):308–328, 2017.

[4] Michael Decker. Location privacy-an overview. In *2008 7th International Conference on Mobile Business*, pages 221–230. IEEE, 2008.

[5] Karel Dhondt, Victor Le Pochat, Alexios Voulimeneas, Wouter Joosen, and Stijn Volckaert. A run a day won't keep the hacker away: Inference attacks on endpoint privacy zones in fitness tracking social networks. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 801–814, 2022.

[6] Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 31–42, 2003.

[7] Wajih Ul Hassan, Saad Hussain, and Adam Bates. Analysis of privacy protections in fitness tracking social networks-or-you can run, but can you hide? In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 497–512, 2018.

[8] Ricardo Mendes, Mariana Cunha, and João P Vilela. Impact of frequency of location reports on the privacy level of geo-indistinguishability. *Proc. Priv. Enhancing Technol.*, 2020(2):379–396, 2020.

[9] Reza Shokri, George Theodorakopoulos, Carmela Troncoso, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. Protecting location privacy: optimal strategy against localization attacks. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 617–627, 2012.

Poster: Towards a Secure and Practical System to Obfuscate Tor Network Traffic

Minjae Seo[†]*, Myoungsung You[†]*, Taejune Park[‡], Seungwon Shin[†], and Jinwoo Kim[§]**
[†]*KAIST,* [‡]*Chonnam National University,* [§]*Kwangwoon University*

Abstract—**Tor (The Onion Routing) has emerged as a promising open-source privacy network for providing anonymous communication in sensitive tasks, such as journalism and activism. However, it is also susceptible to deanonymization attacks, particularly flow correlation attacks, which identify users by correlating unique traffic flow characteristics observed at the ingress and egress segments of a Tor connection. To mitigate these attacks, various traffic obfuscation techniques have been developed; however, they often suffer from bandwidth waste and CPU resource exhaustion, critical issues in the Tor ecosystem where efficient utilization of the limited, voluntarily-shared bandwidth and resource of nodes is crucial. In this paper, we propose `ODIN`, a Tor-tailored hardware-based security solution that hinders adversaries from launching flow correlation attacks. Unlike previous approaches that primarily added randomly crafted padding bytes to packets, `ODIN` innovatively minimizes bandwidth waste by slicing and reorganizing packets destined for the same server with the aid of the context of Tor circuits, effectively addressing the issue of wasted network resources.**

1. Introduction

Tor (The Onion Routing) is an open-source privacy network that enables Internet users, particularly those in contexts with active censorship such as journalists, activists, or whistle-blowers, to access anonymous online services. The Tor network is widely used, with over 3 million daily users and a total of 6,000 volunteer relay nodes transmitting terabytes of traffic every day.

As the nature of Tor aims to facilitate anonymous communication for sensitive tasks, Tor is naturally a target for many deanonymization attacks. Among them, the *flow correlation attack* [8] is obviously the most powerful attack correlating unique characteristics of traffic flows (e.g., packet sizes, interval times) observed from the ingress and egress segments in a Tor connection. The pioneering research shows that it is effective for deanonymizing Tor clients and servers. For example, RAPTOR [10] achieved a 90% accuracy in deanonymizing user identity. Also, the state-of-the-art flow correlation of DeepCorr [9] offered a correlation accuracy of 96% even with shorter flow observations than previous methods required.

In recognition of the threat posed by flow correlation attacks, several *traffic obfuscation* techniques [3], [4], [6] have been developed to mitigate them. However, these techniques are not well suited to the requirements of the Tor network due to the following issues:

CPU Resource Exhaustion. Existing secret Tor relay nodes (e.g., bridge nodes [4]) or pluggable transports (e.g., obfs4, meek [6]) are utilized to conceal the fact that clients are using the Tor network by randomizing packet bytes or disguising a Tor connection as other legitimate connections (e.g., Skype). However, these techniques rely on software-based systems to process individual packets, which significantly consumes CPU resources compared to using ordinary Tor relay nodes. Among them, it is particularly crucial for exit nodes to ensure efficient resource utilization, as they play a vital role in maintaining uninterrupted routing and communication in the Tor network.

Tor Bandwidth Waste. Other techniques [5], [7] add crafted padding bytes to the original packets and control packet transmission timing to obfuscate the size and interval time of individual packets. However, the inclusion of padding bytes in these methods imposes additional burdens on individual packets and can waste Tor network bandwidth, particularly in situations where a limited number of volunteer nodes are shared by many Tor clients. As a result, most existing obfuscation methods are not well adapted to be used practically in the real Tor network.

In this paper, we propose `ODIN`, a hardware-based security solution that addresses both issues raised by existing solutions. To conserve CPU resources, `ODIN` offloads all Tor traffic obfuscation logic to a programmable hardware network interface card (i.e., SmartNIC). Also, `ODIN` addresses the issue of bandwidth waste by slicing and reorganizing packets destined for the same server with the aid of the context of Tor circuits. This approach allows `ODIN` to obfuscate the original traffic pattern by assembling two distinct flows, significantly saving bandwidth and CPU resources compared to existing solutions.

We implement a full prototype of `ODIN` using the NVIDIA BlueField 2® SmartNIC [2] and conduct extensive experiments within a realistic Tor testbed environment. In our private testbed, we establish a trusted exit node, `ODIN`, responsible for transmitting Tor traffic to the server. Additionally, we implement a receiver program, utilizing recent kernel networking features, designed to operate on the destination server. The receiver program ensures that Tor services maintain transparency when deploying `ODIN` by restoring the combined packets received from `ODIN`. In our evaluation, we primarily demonstrate the effectiveness of obfuscation and performance improvements by presenting different aspects of traffic flow after the deployment of `ODIN`.

2. Technical Background and Preliminaries

2.1. Tor Circuit

Tor works by leveraging a core technology known as *onion routing*. Onion routing transmits encrypted data (in multiple layers of encryption) to the final destination (i.e.,

*Co-first authors, **Corresponding author* server) through a minimum of **three** types of relay nodes in the Tor network.

Entry node, also known as the guard node, serves as the initial *ingress* point in the Tor network, receiving connections directly from Tor clients. Its main role is to receive multi-layer encrypted traffic from clients, decrypt the outermost layer, and forward the partially decrypted traffic to the next node in the Tor network. It is worthy note that while the entry node has access to the client's *identity* (e.g., IP address) and can detect the client's utilization of Tor, it is unable to decrypt the final destination or the content of the client's traffic due to the multi-layer encryption mechanisms inherent in onion routing.

Middle node is the second node to which the Tor client connects and acts as a linking chain. It serves a crucial role by stripping the second layer of encryption and seamlessly routing Tor traffic between the entry and exit nodes. Crucially, due to the principles of onion routing, this middle node is unable to discern the identities or information of either the client or the server.

Exit node is the last *egress* point where Tor traffic finally hits the public Internet. It communicates directly with the server, which is the final destination, making it capable of identifying the server's true identity and decrypting the encrypted layers of Tor traffic. As a result, the exit node becomes a desirable target for several entities (e.g., law enforcement agencies and threat actors) to intercept Tor traffic in the middle.

Given its significance, note that several reputable organizations and privacy-conscious entities already operate *trusted exit nodes* [1] with enhanced security features and capabilities. These trusted exit nodes are specifically designed to protect both the Tor network and service operators from potential hazards and threats associated with exit nodes, thereby enhancing the overall security and integrity of the Tor ecosystem.

2.2. Flow Correlation Attack

In most scenarios of a flow correlation attack, adversaries attempt to link Tor traffic observed from the two points, *ingress* and *egress*. The two points, carrying the information of the real *identity* (e.g., client or server), can be an attractive target in order for adversaries to eavesdrop on Tor traffic in the middle. If adversaries collect enough traffic from both points, they can have the capability of performing a flow correlation attack with following pioneering traffic analyses tailored to Tor network.

Statistical Metric. Several studies have used a statistical metric to measure the flow similarity of traversed flows observed from both *ingress* and *egress* points. For example, Sun et al. [10] used the Spearman correlation coefficient, which centers on a nonparametric measure that can evaluate the statistical dependence between the rankings of two given variables. They extracted the TCP sequence and acknowledgement number from each packet trace to conduct asymmetric correlation analysis that allows adversaries to observe any direction of the Tor traffic at both points (*ingress* and *egress* points). However, one obvious *problem* arises from the fact that they consider the case where there is a long-lasting Tor connection (inevitably needs a long flow observation) rather than aims to consider an intermittent Tor connection which is pervasive in the real world Tor network.

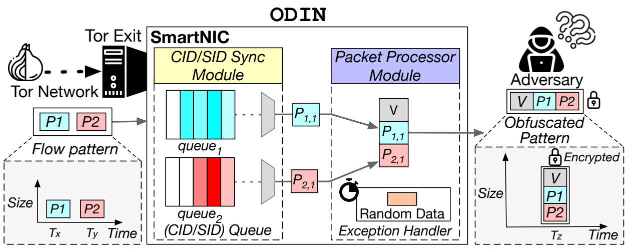


Figure 1: `ODIN` combines packets destined for the same server, thereby rendering the pattern of Tor traffic indiscernible to adversaries.

Deep Learning. To address the issue derived from the statistical metric, a new approach has emerged to measure the flow similarity even when considering a short-lived nature of Tor network. For example, Nasr et al. [9] proposed the state-of-the-art deep learning-based flow correlation technique, DeepCorr [9], that is able to correlate Tor traffic with high accuracy using very short-term observations of Tor connections. Due to the content encryption for each packet, they leveraged an advanced deep learning model in order to learn hidden patterns of flow-level features in Tor traffic. Specifically, they found dominant characteristic in the feature of packet *sizes* and *timings* when correlating Tor traffic. Through extensive experiments, they showed the dangers posed by exposure of flow-level features to adversaries. Thus, it is necessary to hinder adversaries from learning unique patterns through these features.

3. `ODIN` Overview

3.1. Threat Model

The goal of adversaries is to deanonymize clients by correlating ingress flows and egress flows. They compare several unique characteristics of Tor traffic (e.g., packet sizes and timings) instead of attempting to decrypt the content of Tor traffic. To achieve their goal, we consider adversaries who have capabilities to eavesdrop on Tor traffic (especially ingress and egress flows) passively.

We believe this scenario is practical in the real world. In recent years, many threat actors are running hundreds of malicious Tor nodes in order to intercept Tor traffic [11]. Adversaries can further leverage BGP hijacking attacks, which becomes increasingly frequent, by a means to redirect the Tor traffic to themselves [10]. There might also be more powerful adversaries, such as governmental agencies, who can perform wiretapping attacks directly on multiple Internet ASes or intercontinental fiber optics. We contemplate the deployment of `ODIN` on a trusted exit node, managed securely by reputable entities, mitigating the risk of adversary compromise.

3.2. System Design

As shown in Figure 1, the design of `ODIN` is primarily aimed at achieving a secure and practical obfuscation system that addresses the following three key aspects:

Context-aware Obfuscation. In order to develop a Tor-tailored obfuscation system, `ODIN` ensures consistency by leveraging the context information of established Tor circuits. As shown in Figure 1, the CID/SID Sync module within `ODIN` perpetually synchronizes with the circuit identifier (CID) and stream identifier (SID) information derived from the Tor application through its default API. The CID provides specificity about the Tor circuit that a connection refers to, whereas the SID identifies distinct TCP flows. Utilizing both CID and SID, ODIN combines distinct flows that are destined for the same server.

Bandwidth-saving Obfuscation. In an environment where a limited number of Tor nodes share their bandwidth, ODIN reduces bandwidth waste effectively. Thus, the packet processor module in ODIN minimizes Tor bandwidth by piggybacking on packets already destined for the same final server. As shown in Figure 1, the packet processor module removes the headers (e.g., Ethernet) of two paired packets, and subsequently, it combines the remaining contents of both packets with a virtual header. The total size of an n-th obfuscated packet P'_n is defined as $P'_n = V + P_{r,n} + P_{r+1,n}$, where $P_{r,n}$ denotes the size of n-th packet in the r-th queue and V denotes the size of the virtual header. This approach effectively obfuscates the traffic pattern by combining two distinct flows into a single flow. After combining, the resulting packets are encapsulated, encrypted, and sent to the destination server.

Upon arrival at the receiving server, the combined packets are first decrypted. Subsequently, utilizing the data embedded within the decrypted virtual header, the receiver program reassembles them back into their original form. This reconstitution of packets is performed prior to the initiation of kernel network stack operations, thereby guaranteeing that the Tor service on the destination server receives the packets irrespective of the obfuscation process conducted by ODIN. The overall bandwidth waste is significantly reduced compared to existing padding-based methods, making ODIN a practical solution in environments where efficient utilization of limited shared bandwidth is essential.

Spatial and Temporal Obfuscation. ODIN operates with obfuscation functionalities for both size and timing under all circumstances. To this end, we consider two scenarios: *(i)* when only a single packet is in the queue or remains after pairs of packets have been processed within the time threshold; and *(ii)* when the combined size of two packets exceeds the MTU size.

To address these scenarios, the packet processor module stores the sizes of previously transmitted packets. Utilizing this information, it calculates the optimal padding size for the remaining packet as follows:

$$padding \leftarrow \textsc{CreateRandomData}(MTU - P_{r,n})$$
$$padingSize \leftarrow \textsc{GetSimilarSize}(L, P_{r,n}, padding)$$

, where MTU is the maximum transmission unit, $P_{r,n}$ is the size of an n-th packet in the r-th queue, and L is the size list of previously transmitted packets. Then, the packet processor module identifies a size that is most similar to those within the list of previously transmitted packets. This selected size is then used to create a packet that is sent to the server, thereby preventing potential adversaries from discerning any unique packet characteristics. This approach guarantees size and timing obfuscation under all conditions while preserving the system's effectiveness.

4. Evaluation

We conduct a preliminary experiment on our private Tor testbed, comprising three Tor container nodes (i.e., entry, middle, and exit) running on two physical machines

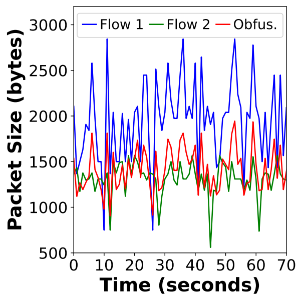


Figure 2: Traffic pattern before/after deploying ODIN.

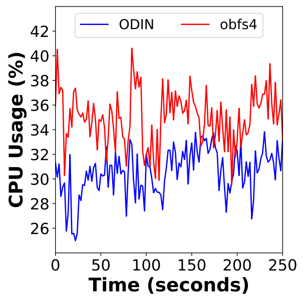


Figure 3: CPU usage of ODIN and obfs4.

with several services (e.g., video streaming and file downloading). The evaluation of ODIN primarily centers on two key aspects: (i) its efficacy in obfuscating Tor traffic and (ii) its ability to impose lower overhead compared to an existing obfuscation solution.

Figure 2 exhibits distinct flow patterns with and without ODIN's obfuscation functionality. ODIN successfully combines two distinct traffic flows, resulting in discernible deviations from the original traffic flow patterns. Figure3 presents the measured CPU usage within a specific time window, comparing the deployment of ODIN and obfs4 for obfuscation purposes. On average, ODIN exhibits a 5% reduction in CPU overhead compared to obfs4.

5. Future Work

In future work, we will assess the resilience of ODIN-obfuscated traffic against advanced flow correlation attacks, including those using deep learning techniques.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government. (MSIT) (No. RS-2022-00166401, 2022R1C1C1006967)

References

[1] Tor Exit Enclave. https://help.duckduckgo.com/duckduckgo-help-pages/privacy/tor-exit-enclave/, 2022.

[2] NVIDIA BLUEFIELD-2 DPU, 2023. https://resources.nvidia.com/en-us-accelerated-networking-resource-library/bluefield-2-dpu-datasheet.

[3] Tor Circumvention, 2023. https://tb-manual.torproject.org/circumvention/.

[4] Types of Relays on the Tor Network, 2023. https://community.torproject.org/relay/types-of-relays/.

[5] Kevin P Dyer et al. Peek-a-boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *IEEE Symposium on Security and Privacy*. IEEE, 2012.

[6] David Fifield et al. Blocking-resistant Communication Through Domain Fronting. *Proc. Priv. Enhancing Technol.*, 2015(2):46–64, 2015.

[7] Roland Meier et al. ditto: WAN Traffic Obfuscation at Line Rate. In *NDSS Symposium*, 2022.

[8] Steven J Murdoch and George Danezis. Low-cost Traffic Analysis of Tor. In *IEEE Symposium on Security and Privacy*, 2005.

[9] Milad Nasr et al. DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning. In *ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[10] Yixin Sun et al. RAPTOR: Routing Attacks on Privacy in Tor. In *24th USENIX Security Symposium*, 2015.

[11] Matthew K Wright et al. An Analysis of the Degradation of Anonymous Protocols. In *NDSS Symposium*, 2002.

Poster: Using Causal Inference to Extract Hidden Information from Dependency Modelling

Ayodeji Rotibi
Computer Science and Informatics
Cardiff University
Cardiff, United Kingdom
rotibiao@cardiff.ac.uk

Neetesh Saxena
Computer Science and Informatics
Cardiff University
Cardiff, United Kingdom
saxenan4@cardiff.ac.uk

Pete Burnap
Computer Science and Informatics
Cardiff University
Cardiff, United Kingdom
burnapp@cardiff.ac.uk

***Abstract*—Dependency Modelling is an established Probabilistic Risk Analysis method that is frequently used to identify and quantify cyber risks in complex environments, such as Industrial Control Systems. The method is useful for examining the interrelationships between different variables, but the limited data exposure in the modelling restricts its ability to analyse multiple independent variables simultaneously or sequentially. In response to this limitation, we present a new technique that leverages the Bayesian Network method to draw inferences from unrelated events and uncovers hidden insights that Dependency Modelling may overlook. We conducted an evaluation of our proposed technique using lab-generated data that mimics Colonial pipeline operations. Our results demonstrated that the proposed technique exposes previously undetected aspects of the dependency model, providing business and asset owners with a more comprehensive understanding of their cyber risks and facilitating better decision-making. Our technique represents a significant advancement and is the first to apply this inference method to Dependency Modelling.**

***Index Terms*—Cyber risks, Dependency Modelling, Bayesian Network, Variable Elimination**

I. Introduction

The industrial technology landscape is continually evolving, resulting in an increased connection of processes and components that enhance productivity and bottom-line impact. However, this transformation also brings new risks to operational technology (OT) systems and operations, increasing complexity and posing significant cybersecurity challenges [1].

Despite continuous efforts by industries and governments to enhance cybersecurity, major industrial cyber breaches remain as likely today as they did ten years ago. Recent cyber attacks on Colonial Pipeline and JBS Foods have highlighted the consequences of cyber threats and the vulnerabilities of exchanging data and dependencies in enterprise systems [2]. Successful attacks can lead to a complete system failure, emphasising the need to evaluate alternative approaches to mitigate cyber risks in complex systems.

Dependency Modelling (DM) provides a comprehensive framework for establishing links between system events, processes, and dependencies, enabling accurate risk assessments to support informed decision-making and enhance cybersecurity [3]. Despite its capabilities, DM's limitations prevent it from providing sufficient insights to fully understand a system's complexity beyond conventional approaches, highlighting the need for alternative methods.

This work has been supported by the Knowledge Economy Skills Scholarships (KESS2) - a major pan-Wales operation supported by European Social Funds (ESF) through the Welsh Government, and Thales UK

Contribution: Our proposed technique introduces causal inference into Dependency Modelling (DM) which allows the analysis of multiple independent nodes and accounting for simultaneous or sequential changes within the model. This multi-nodal analysis increases the identification of cyber risks that are synonymous with the tight coupling characteristics phenomena in complex systems where multiple events can fail synchronously. We believe this enhancement positions DM as a preferred method to identify cyber risks in complex environments, including Industrial Control Systems (ICS).

II. Related Work

The potential of Bayesian Networks (BN) as an adaptable and effective tool in handling incomplete or uncertain information has been recognised by researchers [4]. Previous studies, such as [5], [6], have demonstrated the suitability of BN in detecting intrusion and insider threats in system networks, showcasing its efficiency in mitigating cyber risks.

However, the use of BN in identifying cyber risks within large and complex systems, including ICS, remains limited. Existing research does not account for the impact of simultaneous or sequential failure within complex system networks, nor have effective techniques for enhancing cyber risk identification in such systems been proposed.

III. Approach

Our approach utilises Directed Acyclic Graphs (DAGs), which model the conditional dependencies between variables in a probabilistic model. We implement Bayesian Networks (BN) to perform statistical inference on DM and calculate the *conditional probabilities* of unknown variables based on their observed values. While both BN and DM are usually causally constructed, BN assumes that most variables are independent of their preceding variables, whereas DM assumes that all variables may be directly impacted by their predecessors. This property makes BN advantageous, enabling the identification of a subset of preceding parameters for each parameter in turn, which allows us to use Variable Elimination (VE) for causal inference to identify hidden or previously unknown risks within the system [7].

The VE probabilistic inference algorithm calculates the marginal probabilities of a target variable by recursively eliminating irrelevant network variables that do not impact the target variable. This efficiency in handling large and complex BN makes VE the preferred algorithm over others like the Junction Tree (JT) and Monte Carlo Markov Chain.

To retrieve hidden data from the model, we construct an inference query in the form of $P(Y|E = e)$, where Y and E are disjoint variables in the model, and E is an observed variable with a value of e [7].

IV. Validation

We aim to determine if the causal inference technique can reveal changes in the model's sensitivity when considering the combination of multiple independent nodes. The traditional 3-Point Sensitivity (3PS) approach used in DM can only assess the sensitivity impact of a single *leaf* node at a time.

To validate our approach, we employed a case study that mimics Colonial pipeline operations in an Industrial Control System (ICS) environment, named PipelineX. We focused on the communication between the IT and OT networks involved in the shipping process to track product delivery. After receiving an order from a customer, an operator at the enterprise network verifies product availability via the production network before initiating the shipping process. The shipping process generates a trigger to load the product for delivery. Figure 1 illustrates the business process description. Additional information includes the following:

- Remote login to the IT network is available via a secured Virtual Private Network (VPN) infrastructure, managed by the Enterprise Access Control.
- There is no network segmentation infrastructure between the IT and the OT networks.
- A loss of availability on the IT network due to an attack could disrupt production on the OT network.

Our data is an adaptation from an existing manufacturing environment with 67 nodes in the model. Each node has three attributes: name, dependencies (name of parent node), and the percentage probability of being in a desired state. Each node is numbered (ref) from 0 (the goal/root node) to 66. We have included some node names and descriptions in Table I.

TABLE I
NODE WITH REFERENCE NUMBER

Ref	**Node Name**	Description
0	Secure and Safe Production	This is the goal of the business
9	Enterprise Access Control	Access Policies are implemented
34	Wireless Protocols	Protocols are updated and secured
40	Background Checks	Security check conducted on users
41	Roles and Responsibilities	Clearly defined and assigned
42	Training	Appropriate training conducted
43	Specialised Training	Training specific to functions
44	Security Awareness	Basic requirements for users
45	Security Responsibilities	Assigned and owned
46	Event and Incident Mgt	Logged and reviewed

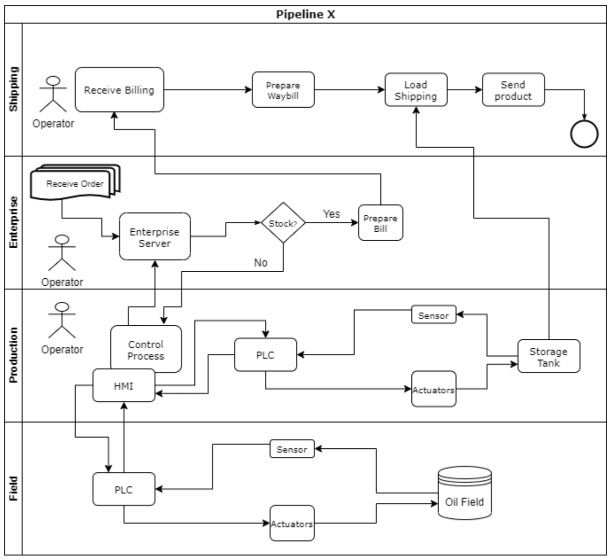


Fig. 1. Shipping Process Flow in Pipeline X

Conventional behaviour expectation dictates that if a node fails due to an event, its probability is set to 0 (or 0%). This event results in a negative impact on the overall model. Conversely, setting a node to 1 (or 100%) due to an improvement in a certain event positively impacts the model. To identify the node with the highest impact (sensitivity), we set each node to 0 and 1, respectively, and performed causal inference to obtain new probability values for the root node. This process is repeated with combinations of two and three nodes.

The resulting sensitivity scores are presented in Tables II, III, and IV, where each table lists the top-5 sensitivity scores. The *Node* column indicates the node number, corresponding to its name in Table I. The *Probability* column displays the current marginal probability of the overall goal. The last two columns reveal the sensitivity values, representing the difference between the marginal probability and the computed probability when the node is turned off (E=0) and when it is turned fully on (E=1). As an example, in Table II, Row 1 Column *E=0* displays the result of 0.16361779 - 0.042172706, while Column *E=1* is derived from *E=1*, i.e 0.167503027 - 0.16361779.

TABLE II
CAUSAL INFERENCE FOR SINGLE EVENT

Node	Probability	E=0	E=1
[40]	0.16361779	0.121445081	0.00388524
[41]	0.16361779	0.121445081	0.00388524
[43]	0.16361779	0.107265991	0.003431626
[44]	0.16361779	0.107265991	0.003431626
[45]	0.16361779	0.107265991	0.003431626

Figure 2A, B and C show the 3PS plots for each table. A short bar indicates low sensitivity, while a longer bar represents higher sensitivity. The colour of each bar corresponds to its influence, where red-coloured bars suggest a negative impact and green-coloured bars exhibit a positive influence. The junction between the bars indicates the sensitivity level

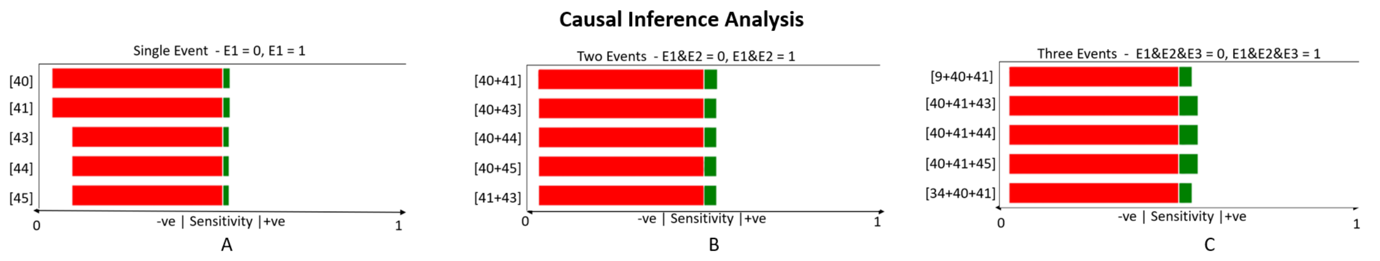


Fig. 2. 3-Point Sensitivity Using Causal Inference Analysis

TABLE III
CAUSAL INFERENCE FOR TWO (COMBINED) EVENTS

Node	Probability	E1=E2=0	E1=E2=1
[40+41]	0.163617787	0.125440583	0.007890686
[40+43]	0.163617787	0.124978108	0.007423034
[40+44]	0.163617787	0.124978108	0.007423034
[40+45]	0.163617787	0.124978108	0.007423034
[41+43]	0.163617787	0.124978108	0.007423034

TABLE IV
CAUSAL INFERENCE FOR THREE (COMBINED) EVENTS

Node	Probability	E1=E2=E3=0	E1=E2=E3=1
[9+40+41]	0.163617787	0.125560776	0.007904915
[40+41+43]	0.163617787	0.125559031	0.011537933
[40+41+44]	0.163617787	0.125559031	0.011537933
[40+41+45]	0.163617787	0.125559031	0.011537933
[34+40+41]	0.163617787	0.125553184	0.007904016

concerning the overall goal or how far it is from the probability of the goal. We observed that setting the probabilities of three nodes to zero (E1 = E2 = E3 = 0) resulted in longer red bars than only setting the probabilities of two nodes to zero (PE1 = E2 = 0). This indicates that the model is more sensitive, with a higher negative impact with more nodes. Conversely, setting the probabilities of three nodes to one (E1 = E2 = E3 = 1) resulted in a higher positive influence, with longer green bars than only setting the probabilities of two nodes to one (E1 = E2 = 1).

From the information obtained in the model, we conducted a frequency analysis to identify the nodes with the most influence, which is a function of how many times they occur in combination with other nodes. As shown in Figure 3A, Node 41 is the most influential when we perform a two-nodal causal inference. However, in a three-nodal causal inference, nodes 40 and 41 are of equal influence as both of them appeared five times, as shown in Figure 3B. The interpretation is that the asset owner may want to pay closer attention to these nodes.

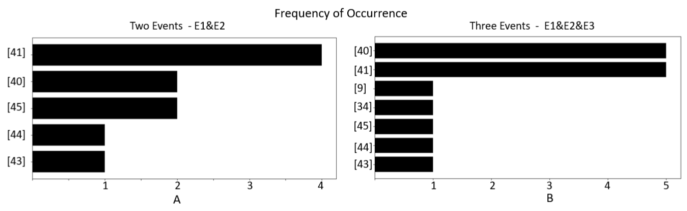


Fig. 3. Node Frequency Analysis

We validated our technique by checking the consistency of the sensitivity pattern among the three graphs. As the number of nodes in the causal inference calculation increases, the bars in both directions 3PS become longer, indicating an increased sensitivity (impact) and decreased probability of success if multiple nodes fail simultaneously. From Tables III and IV, we observed that node 40 is more critical in a 2-nodal inference while node 41 is most critical in a 3-nodal inference. This suggests that our technique can uncover critical nodes that could potentially prevent attacks, as shown by the case of the Colonial Pipeline cyber attack. Our technique enables us to discover more information from DM than it currently provides, increasing the potential for system owners to proactively manage and mitigate risks.

V. CHALLENGES AND FUTURE WORK

BN learning requires extensive computation to process causal queries for two or more nodal combinations, creating scalability issues for larger models with many nodes. To overcome these challenges, our consideration is limited to a system-driven model with a focus on processes and the interaction between processes. In the future, we hope to leverage DM's new capacity to model complex systems and to develop predictive models that can forecast future cyber risk trends, based on past data.

REFERENCES

[1] S. McLaughlin et al., 'The cybersecurity landscape in industrial control systems', Proceedings of the IEEE, vol. 104, no. 5, pp. 1039–1057, 2016.
[2] S. M. Kerner, "Colonial pipeline hack explained: Everything you need to know," WhatIs.com, https://tinyurl.com/393s2m2j (accessed Jan 26, 2023).
[3] A. O. Rotibi, N. Saxena, P. Burnap, and A. Tarter, 'Extended Dependency Modeling Technique for Cyber Risk Identification in ICS', IEEE Access, vol. 11, pp. 37229–37242, 2023.
[4] D. Koller and N. Friedman, Probabilistic graphical models: principles and techniques. MIT press, 2009.
[5] P. G. Bringas, 'Intensive use of Bayesian belief networks for the unified, flexible and adaptable analysis of misuses and anomalies in network intrusion detection and prevention systems', in 18th International Workshop on Database and Expert Systems Applications (DEXA 2007), 2007, pp. 365–371.
[6] A. Wall and I. Agrafiotis, 'A Bayesian approach to insider threat detection', Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, vol. 12, no. 2, 2021.
[7] C. J. Butz, W. Yan, P. Lingras, and Y. Y. Yao, 'The CPT Structure of Variable Elimination in Discrete Bayesian Networks', in Advances in Intelligent Information Systems, Z. W. Ras and L.-S. Tsay, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 245–257.

Poster: Don’t Trust, Verify: Empowering Last-Mile Security and Privacy in Web3

Weihong Wang
imec-DistriNet, KU Leuven
Leuven, Belgium
weihong.wang@kuleuven.be

Tom Van Cutsem
imec-DistriNet, KU Leuven
Leuven, Belgium
tom.vancutsem@kuleuven.be

***Abstract*—The bridging of Web2 and Web3 infrastructure is a crucial step towards a fully decentralized internet. However, participating in blockchain systems can be a significant barrier for most users due to factors such as the high cost of operation and the computational resources required. To interact with the blockchain, Web3 application builders rely on endpoint API services, which can cause centralization issues in terms of privacy, availability, and security. In response, the community has turned to light nodes as a potential solution for resource-constrained clients such as smartphones and browsers to facilitate involvement. In this paper, we provide an overview of recent studies in light nodes, along with an analysis of light client implementations in Ethereum and their improvements. We conclude with a call for further development of light node technology to enable broader and more diverse participation in decentralized systems.**

***Index Terms*—Web3, Blockchain networks, Light clients**

I. Introduction

Web3, also known as the Decentralized Web, is a vision for the next generation of Web applications where users can have stronger ownership over their data and identity, supported by blockchain technology. However, transitioning to Web3 from the current web (Web2) poses several challenges [1]. Many low-capacity remote clients, such as mobile wallets, lack the ability to run a full blockchain node due to the computational and storage requirements involved in downloading and verifying all transactions recorded by the chain on the first-time bootstrapping process. As a result, users have increasingly relied on trusted intermediaries, such as Infura and Alchemy. These companies offer JSON-RPC services, which is a remote procedure call (RPC) protocol that utilizes JavaScript Object Notation (JSON) to serialize data and send requests to a server. By using these services, users can access blockchain networks with reduced friction.

However, the reliance on intermediaries creates multiple challenges that can negatively impact the **privacy**, **availability**, and **security** goals of Web3 [2]. Specifically, intermediaries have unrestricted access to transaction requests and sensitive data from end-users, including IP addresses, which can jeopardize user privacy. Moreover, intermediaries have the ability to control the content displayed on the blockchain [1], which can affect the availability of data and undermine its reliability. Finally, there is no guarantee that the data returned by intermediaries is accurate or tamper-proof, which can compromise the security of transactions and user assets.

While using trusted intermediaries can provide users with a better user experience, it is essential to find solutions that provide an optimal balance between minimizing user hardware requirements and maintaining privacy and security.

In this paper, we focus on the current infrastructure that bridges Web2 and Web3 from the perspective of decentralized applications (DApps) builders. As builders are responsible for providing end-users with access to blockchain content and interactions, it’s crucial that we understand the privacy, availability and security implications of current approaches. Our long-term goal is to make access to Web3 seamless for Web clients while minimizing the trust assumptions.

II. Problem statement

In this section, we will discuss the current infrastructure for bridging Web2 and Web3 and discuss in detail how it can cause privacy concerns, availability challenges, and security threats.

A. Current Infrastructure for Bridging Web2 and Web3

Payment-based. Wallets play a critical role in Web3 by providing a secure way to manage blockchain assets and facilitating interactions with decentralized applications. To obtain the current token balance, wallets use JSON-RPC APIs to send requests to the endpoints of full nodes, which allows them to communicate with the blockchain network and track the tip of the chain. For example, MetaMask is a widely-used Web3 wallet for Ethereum that retrieves the user’s account address and uses Infura [3] as its default endpoint provider to query the Ethereum blockchain and obtain the balance for that address.

Application-based. When an end-user wants to interact with smart contracts on the blockchain through a website, a library such as Ethers.js or Web3.js is required to facilitate communication with the blockchain from the remote client. Ethers.js, for example, provides a *Provider* object which enables developers to query the blockchain and get information. Additionally, a *Signer* object is used to execute state-changing operations by signing transactions with the user’s private key, which is always stored in the wallet, and then broadcasting them to the network via the *Provider* object. A *Provider* object can be injected by a browser extension wallet like MetaMask, or backed by an RPC API service available in all node implementations. Because running a full node locally is costly, third-party web services such as Infura and Alchemy are widely used by developers to provide access to the blockchain network.

B. Privacy, Availability and Security Concerns

Privacy. When using payment-based wallets, users often share their account address with third-party endpoint providers, posing privacy risks if their identity is linked to the address, especially for valuable accounts with significant cryptocurrencies.

In application-based scenarios, external API services typically require developers to obtain project-specific API keys through registration and acceptance of terms of service from providers [4]. It raises concerns about privacy if the provider collects and analyzes the usage patterns and behaviour of the clients for advertising or other purposes.

Availability. Depending solely on a single provider for Web3 interactions can create a single point of failure, posing risks in case of provider issues like malicious attacks or network congestion. This undermines the resilience and objective of Web3 to withstand attacks and outages.

Furthermore, if the provider is not supportive of certain content on the blockchain, it can result in censorship. It is particularly problematic in terms of a more transparent and open Web3 world, where censorship should not be the sole decision of a small group of people.

Security. Third-party intermediaries can potentially provide incorrect or fraudulent data if there is no verification process in place. This can occur if the intermediaries lack sufficient incentives to behave honestly. Additionally, end-users need accessible proof to verify that the data they receive is up-to-date and authentic.

III. Light Nodes

A. Overview of Light Nodes Schemes

The immutability of a blockchain is maintained through the linked list of blocks, known as the distributed ledger, in which each block contains information from the previous block and is created through a consensus algorithm such as Proof of Work using hash puzzles or Proof of Stake based on a proposal system. Therefore, the process of efficiently rebuilding and verifying the entire ledger is crucial for new participants in achieving agreement on the latest state of the blockchain.

There are two kinds of nodes that a user can choose to interact with a blockchain network: *full nodes*, which store and validate the complete historical ledger of transactions, and *light nodes*, which rely on full nodes and solely verify block headers (i.e., small, unique cryptographic fingerprints) with Merkle tree-based proof system to reduce the amount of data that needs to be processed. Light nodes, such as the concepts of Simplified Payment Verification (SPV) for Bitcoin [5] and light nodes based on Proof of Stake (PoS) in Ethereum [6], work as a solution for lightweight clients like smartphones and browsers to avoid intensive storage and computation resources required by a full node software. The streamlined process to participate in a blockchain network by light nodes is an option to achieve a more diverse node composition.

To further address the computational challenges associated with the linear growth of blockchain size in light nodes, several studies have explored solutions for making proofs logarithmic such as super-light nodes that only check a subset of block headers. For Proof of Work protocols, PoPoW [7] reduces the linear complexity to be logarithmic with a notion of ”superblock”. NIPoPoW [8] makes it non-interactive, giving the possibility to interact with multiple servers. FlyClient [9] further supports checking variant working difficulty by Merkle Mountain Range (MMR) and sampling. Mina [10] and PoNW [11] propose recursively composed proofs with SNARKs to achieve constant bootstrapping complexity. [12] is the first succinct proofs of proof-of-stake (PoPoS) protocol, taking the form of a Merkle tree of PoS epochs and a bisection game design. BITE [13] and DCert [14] are working on using trusted execution in lightweight clients.

B. Light Node Implementations in Ethereum

The state of light client implementations in major blockchain systems is presented in detail in [15]. However, with the transition of Ethereum from PoW to PoS through the merge [16], a new “sync committee” is introduced in Ethereum 2.0 to facilitate the proposal of the new head of the chain by signing proof commitments and handover signatures. It enables the near-term implementations of Ethereum light nodes which is a topic not covered in [15].

Before the merge, Ethereum was using the beacon chain as the consensus layer while the previous main chain acted as the execution layer. After the merge, we now have one single PoS Ethereum chain. Therefore, an execution client such as Geth (the Go implementation of Ethereum) needs to be coupled to a consensus client like Nimbus (the Nim implementation of Ethereum 2.0) in terms of running a node. Presently, several light clients including execution, consensus, and combined execution/consensus light clients are in development. While Lodestar [17] and Nimbus [18] are consensus light client implementations, Helios [19] is a combined execution and consensus light client developed in Rust. In the next subsection, we will discuss the security measures taken by Helios.

Kevlar [20], based on the research work [12], is a CLI tool for RPC Proxy fully compatible with PoS Ethereum and achieves an asymptotically logarithmic bootstrapping complexity, making it highly efficient.

We summarize the existing studies on light clients and corresponding implementation projects in Table I.

C. Improvements on privacy, availability and security

Decentralized RPC requests. When building DApps with the Ethers.js library, one useful feature is its ability to link to multiple endpoint providers. This is made possible through the library’s default Provider object, where you can configure several API keys for your network and verify their results internally. Additionally, DRPC and Pocket are optional decentralized networks of RPC endpoints. This feature is particularly important for light nodes, which requires the ability to

TABLE I: Overview of light client schemes and implementations

Scheme	Consensus	Complexity	Infra Compatibility	Crypto Primitives	Security assumptions	Implementations
SPV [5]	Any	Linear	Any	Merkle root hash	-	-
Ethereum 2.0 [21]	PoS	Linear	Fully compatible	Merkle root hash	-	Nimbus, Helios, Lodestar
NIPoPoW [8]	PoW	Sublinear	Modification	NIPoPoWs	Fixed difficulty	Ergo, WebDollar, Nimiq 1.0
FlyClient [9]	PoW	Sublinear	Modification	MMR commitments	-	ZCash
PoPoS [12]	PoS	Sublinear	Fully compatible	Merkle root hash	-	Kevlar
PoNW [11]	PoW	O(1)	Modification	SNARKs	Trusted setup	-
Mina [10]	PoS	O(1)	New System	SNARKs	Trusted setup	Mina
DCert [14]	Any	O(1)	Any	-	Trusted Execution	DCert

connect to different full nodes to ensure high availability and reliability in third-party services, such as when bootstrapping and verifying transaction inclusion. This feature allows light nodes to maintain a consistent connection to the network, even if one full node becomes unavailable or unreliable.

RPC Method for Retrieving Merkle Proofs. The widely supported *eth_getProof* RPC method, proposed in EIP-1186 [22], provides Merkle proofs for specific transactions or storage data in a given block. After retrieving block headers from full nodes and verifying their validity based on protocol rules, a light node on Ethereum can easily verify the block root hash value and Merkle proof. For example, Helios uses a locally-run proxy server to proxy requests to the execution endpoint and obtain block proof, and then provides validated API responses by verifying against the data server using the consensus endpoint. This method offers a secure and efficient approach for light nodes to verify blockchain-stored data.

There are still open problems in light nodes including scalability, caching mechanisms, interoperability, and difficulties posed by new use cases like IoT. Addressing these challenges is crucial for the continued growth and adoption of light clients in Ethereum and other blockchain ecosystems.

IV. Conclusion

The current infrastructure for bridging Web2 and Web3 is facing centralization issues due to the reliance on third-party providers. This violates the expectations of privacy, availability, and security guarantees, which are fundamental to the decentralization ethos of Web3. The use of light nodes provides a potential solution to these issues and has been studied under different protocol rules, with Ethereum being a prominent example. The implementation of light client technology in Ethereum has shown improvements, but there is still a need to find a balance between user hardware requirements and decentralized participation for future development. Ultimately, addressing these issues will be critical for achieving a more decentralized and trustworthy Web3 ecosystem.

Acknowledgment

This research is partially funded by the Research Fund KU Leuven.

References

[1] M. Marlinspike. (2022) My first impressions of web3. [Online]. Available: https://moxie.org/2022/01/07/web3-first-impressions.html

[2] Z. Luo, R. Murukutla, and A. Kate, “Last mile of blockchains: Rpc and node-as-a-service,” *arXiv preprint arXiv:2212.03383*, 2022.

[3] M. Support. (2023) What is infura, and why does metamask use it? [Online]. Available: https://support.metamask.io/hc/en-us/articles/4417315392795-What-is-Infura-and-why-does-MetaMask-use-it-

[4] ConsenSys. (2022) Privacy policy by consensys. [Online]. Available: https://consensys.net/privacy-policy/

[5] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” May 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[6] E. Foundation. (2023) nodes and clients - light node. [Online]. Available: https://ethereum.org/en/developers/docs/nodes-and-clients/#light-node

[7] A. Kiayias, N. Lamprou, and A.-P. Stouka, “Proofs of proofs of work with sublinear complexity,” in *Financial Cryptography and Data Security: FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers 20*. Springer, 2016, pp. 61–78.

[8] A. Kiayias, A. Miller, and D. Zindros, “Non-interactive proofs of proof-of-work,” in *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*. Springer, 2020, pp. 505–522.

[9] B. Bünz, L. Kiffer, L. Luu, and M. Zamani, “Flyclient: Super-light clients for cryptocurrencies,” in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 928–946.

[10] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro, “Coda: Decentralized cryptocurrency at scale,” *Cryptology ePrint Archive*, 2020.

[11] A. Kattis and J. Bonneau, “Proof of necessary work: Succinct state verification with fairness guarantees,” *Cryptology ePrint Archive*, 2020.

[12] S. Agrawal, J. Neu, E. N. Tas, and D. Zindros, “Proofs of proof-of-stake with sublinear complexity,” Cryptology ePrint Archive, Paper 2022/1642, 2022, https://eprint.iacr.org/2022/1642. [Online]. Available: https://eprint.iacr.org/2022/1642

[13] S. Matetic, K. Wüst, M. Schneider, K. Kostiainen, G. Karame, and S. Capkun, “Bite: Bitcoin lightweight client privacy using trusted execution.” in *USENIX Security Symposium*, 2019, pp. 783–800.

[14] Y. Ji, C. Xu, C. Zhang, and J. Xu, “Dcert: towards secure, efficient, and versatile blockchain light clients,” in *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, 2022, pp. 269–280.

[15] P. Chatzigiannis, F. Baldimtsi, and K. Chalkias, “Sok: Blockchain light clients,” in *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*. Springer, 2022, pp. 615–641.

[16] ethereum.org. (2022) Ethereum roadmap / merge. [Online]. Available: https://ethereum.org/en/roadmap/merge/

[17] ChainSafe. (2023) lodestar. [Online]. Available: https://github.com/ChainSafe/lodestar/tree/unstable/packages/light-client

[18] status.im. (2023) nimbus. [Online]. Available: https://nimbus.guide/el-light-client.html

[19] a16z. (2023) helios. [Online]. Available: https://github.com/a16z/helios

[20] S. Agrawal. (2023) kevlar. [Online]. Available: https://github.com/lightclients/kevlar

[21] eth2book. (2023) Part 2: Technical overview — the building blocks - committee. [Online]. Available: https://eth2book.info/capella/part2/building_blocks/committees/

[22] Ethereum Improvement Proposals. (2018) Eip-1186: Rpc-method to get merkle proofs - eth_getproof. [Online]. Available: https://eips.ethereum.org/EIPS/eip-1186

Poster: Are we reasoning about cloud application vulnerabilities in the right way?

1st Stefano Simonetto
Department of Pervasive Systems
University of Twente
Enschede, The Netherlands
s.simonetto@utwente.nl

2nd Peter Bosch
Department of Pervasive Systems
University of Twente
Enschede, The Netherlands
h.g.p.bosch@utwente.nl

***Abstract*—Enterprises are quickly transitioning to container orchestrators, like Kubernetes, which helps developers and engineers manage a large number of container images, pods, and nodes. However, this new approach does not solve the problem of software vulnerabilities but arguably it makes vulnerability management harder. Most of the time, companies have to deal with thousands of containers in a dynamic environment since they can fail, and be rescheduled in other nodes. All these factors have a great impact on the vulnerability management system because the vulnerabilities and misconfigurations in the system are too many to be manually operated, so we seek a tool to highlight the most dangerous (we need a clear definition of dangerous) to prioritize them.**

This paper wants to emphasize the need for a vulnerability prioritization method and a defense technique improvement.

***Index Terms*—Vulnerability, Prioritization, Container orchestration, Attack kill-chain.**

I. Introduction

Cloud application vulnerabilities have devastating consequences on our digital society, threatening our privacy, finances, and critical infrastructure. CISQ (Consortium for Information and software quality) estimates that the cost of poor software quality in the US has grown to at least $2.41 trillion, but not in similar proportions as seen in 2020. The accumulated software Technical Debt (TD) has grown to roughly $1.52 trillion [2].

In the past few years, the research community proposed sophisticated approaches and techniques to enhance automated security testing and promptly identify vulnerabilities before they can be exploited by malicious attackers.

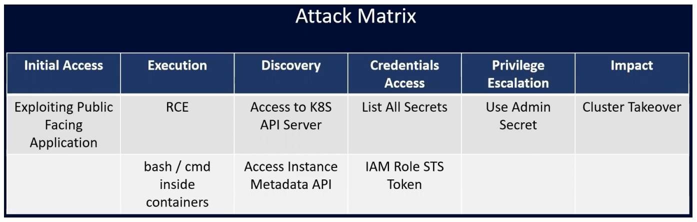
As a result, today a large variety of tools are available to effectively detect vulnerabilities. However, most organizations do not know how to deal with the tons of vulnerabilities also because these tools are prone to produce false positives. The usual behavior is to patch them based on the score produced by each individual vulnerability.

As if the problem wasn't complicated enough, the container orchestration scenario makes the situation more challenging due to the rapid and continuous deployment of new containers and pods in such environments.

In the real world, attackers put together multiple vulnerabilities to successfully compromise systems as shown in Fig.1 which is taken from a real attack scenario and is related to the ATT&CK framework [5].

Thus, analyzing vulnerabilities in combination with each other represents a fundamental step to obtain a realistic "big picture" of their implications.

Furthermore, most defensive solutions are reactive solutions, like intrusion detection systems, system calls monitoring, etc. Even if these techniques are very well-established, they are affected by scalability problems and are not meant to prevent an attack to happen. This poster's abstract aims to create a discussion on how vulnerabilities are managed in the container orchestration environment and particularly in Kubernetes. More precisely, why don't we prioritize the software patches according to the real attack path instead of focusing on a single score? And if this is possible, can we automate this discovering-fixing process? Can we be more proactive during the defense?


Attack Matrix

Initial Access	Execution	Discovery	Credentials Access	Privilege Escalation	Impact
Exploiting Public Facing Application	RCE	Access to K8S API Server	List All Secrets	Use Admin Secret	Cluster Takeover
	bash / cmd inside containers	Access Instance Metadata API	IAM Role STS Token		

Fig. 1. Real attack path in Kubernetes environment

II. Problem statement

Bug-finding approaches have arguably become too successful thanks to:

1) fuzzers which inject automatically semi-random data into a program/stack and detect bugs,
2) scanners that identify vulnerabilities relying on a database of known vulnerabilities.

Industries are finding more vulnerabilities than they can fix promptly. This leads to a known situation in the security domain, named alert fatigue, where security operators cannot stay on top of the large number of alerts produced by potential security issues, as cited by Sysdig in one of their articles [7]:

”Alert fatigue Syndrome is the feeling of becoming desensitized to alerts, causing you to potentially ignore or minimize risks and harming your capability to respond adequately to potential security threats”.
Only a relatively small share of bugs discovered by fuzzers typically has relevant security implications, while the process of identifying and analyzing bugs to generate and deploy patches remains laborious, expensive, and lacks automation. Therefore, the software industry requires a way to set the priority of fixing the most harmful bugs (e.g. stand-alone or concatenated bugs that lead to a dangerous exploit), reducing costs and manual effort for organizations, and minimizing the time window in which users are exposed to potential devastating cyberattacks.
Existing work does not study a critical aspect: how multiple vulnerabilities can be combined. In fact, when taken individually, certain vulnerabilities do not provide highly dangerous capabilities. However, when combined, even low/mid-severity vulnerabilities can result in high-severity consequences.
Thus, approaches that study vulnerabilities individually, according to their severity, are insufficient as they rely on an unrealistic threat model [1].
Another issue in this field is that most of the defense techniques are meant to react during an attack rather than prevent it. In other words, only a few approaches adopt proactive defenses.

III. Discussion and Preliminary results

The main challenge for an effective vulnerability analysis and prioritization strategy is considering multiple vulnerabilities and the capabilities that they enable when combined. In real-world settings, attackers put together (“chain”) multiple vulnerabilities to successfully compromise systems. Thus, ranking vulnerabilities individually, according to their severity, is insufficient and unrealistic.
Most of the work about vulnerabilities kill-chain takes as a baseline the CVSS score which is not a good indicator of how bad the problem is in a particular scenario. For this purpose, the environmental score is introduced but it is implemented in a few papers and when used, most of the time the responsibility is delegated to a security expert. So there is no clear definition of which score is best since we need also to take into account the misconfigurations that are introduced by the infrastructure that allows the programs to work properly, as highlighted in Fig.2.
The literature is also missing a clear definition of what is best to prioritize:

1) single highest CVSS? (We disagree);
2) highest score of a kill-chain? Without taking into account its length;
3) the shortest chain with the highest score?

Finally, there is no clear definition of how to determine the overall kill-chain score. These are critical open problems to be addressed to provide a solid base to automate vulnerability prioritization and patching. The second challenge is changing the defender’s mindset: until today only two big approaches have been proposed in the microservices field about the proactive defense. Moving target defense [3] [4] and Mimic defense [11] leverage the dynamic microservices environment to create uncertainty for attackers, thereby reducing the probability of successful attacks. There are no other real ideas about proactive defenses which instead are highlighted as the leading approaches for the future.

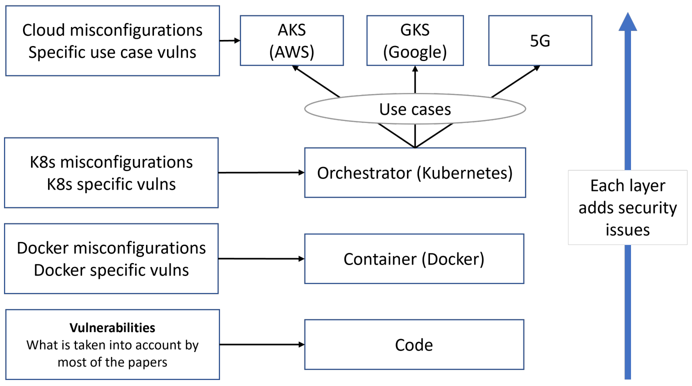


Fig. 2. Each abstraction level adds some vulnerabilities and misconfigurations

IV. Our approach

Our approach to cope with these conceptual problems consist of looking at the scenario from both sides:

1) We want to shed some light on this problem by proposing a method that assigns the proper score to each vulnerability and misconfiguration in a selected environment. We also want to give a clear definition of how to determine the overall kill-chain score and identify the most dangerous ones.
   As soon as it’s done, we aim to automatically suggest patches to fix the problem and automate the process of vulnerability/misconfiguration discovery-fix.
   To give a feeling of the proposed approach:
   a) Discovery: we want to leverage open-source tools like kube-bench [8], kube-hunter [9], and kubeaudit [10] to scan for vulnerabilities/misconfigurations and, based on these findings, start reasoning about the scores and how can they be concatenated.
   b) Fix: this task is very dependent on the problem. For instance, we can try to fix it by applying the new update or release, other times, if the role assigned to a certain user/pod is too permissive, we can try to downgrade it.

   After the patch is applied, start again with this procedure because this last step can lead to a new leak in the system.
   We believe that having a tool that is able to highlight and automatically fix the riskiest path according to the company’s crown jewels, will help to stay on top of the large number of alerts and potential security issues that otherwise could lead to alert fatigue.

2) At the same time, we think taking the right countermeasures against the attackers is very useful since most of the techniques available nowadays reason in a reactive way, e.g. IDS, NIDS, and system call monitoring are all used when the victim is under attack. We are wondering if it is reasonable to create a new proactive defense by probing the scenario before the attacker's action, rather than just responding to it after it has happened. Our long shot is to automate the penetration testing process as much as possible. Even if some tools, like Metasploit [12], already exist and are well-established, we noticed two main gaps:
   a) none of them target specifically the Kubernetes architecture or any other container orchestrator structure;
   b) they provide a framework or general reasoning, leveraging the pen-testers/red-teams knowledge and skills.

   We are not introducing a new concept because penetration testing was done before microservices, but not so often because the business architecture changed a little over time. Today we are faced with a very dynamic and versatile architecture and this change must be managed accordingly introducing, perhaps, a service that answers the question: ”Can I find paths to break the cloud application?”. This is a challenging task because the leading idea is to write a program that is able to automatically attack every single scenario as a pen-tester would do. Since we need to:
   a) gather the pen-testers/hacker behavior and skills;
   b) gather the faulty/vulnerable scenarios;
   c) generalize the techniques for all the possible cases.

   It is easy to realize that the task is very ambitious but, at the same time, it can be a game changer for the security industry.

V. Conclusion

More attention should be granted to this field to reduce the gap between academia and industry in container orchestration security [6]. With this article, we want to emphasize the false feeling of security generated by fixing the individual's most dangerous vulnerability, without taking into account the bigger picture. We need an effective tool that is able to obtain an accurate and comprehensive view of the vulnerabilities, evaluate risks and generate cost-effective patches.

By approaching the problem from the other side, we are thinking about taking the right countermeasures against the attackers by reasoning in a proactive way, trying to make the attack less likely to happen and more challenging for the attackers to execute it.

Finally, to provide a better understanding of our work and to propose our solution in a clear way, we aim to map this flow to the ATT&CK framework [13], which is the most used framework to describe adversary tactics, techniques, and procedures (TTPs).

References

[1] Shemesh D.H. ”Schindel A. KubeCon + CloudNative North America 2022”. (2023, April). [Online]. Available: https://kccncna2022.sched.com/event/182Jl

[2] Herb Krasner. ”NEW RESEARCH: THE COST OF POOR SOFTWARE QUALITY IN THE US: A 2022 REPORT. (2023, April). [Online]. Available: https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report/

[3] Alavizadeh, H., Jang-Jaccard, J., & Kim, D. S. (2018, August). Evaluation for combination of shuffle and diversity on moving target defense strategy for cloud computing. In 2018 17th IEEE international conference on trust, security and privacy in computing and communications/12th IEEE international conference on big data science and engineering (TrustCom/BigDataSE) (pp. 573-578). IEEE.

[4] Jin, H., Li, Z., Zou, D., & Yuan, B. (2019). Dseom: A framework for dynamic security evaluation and optimization of mtd in container-based cloud. IEEE Transactions on Dependable and Secure Computing, 18(3), 1125-1136.

[5] Azarzar O. ”Kubernetes Security Webishop” (2023, April). [Online]. Available: https://resources.lightspin.io/kubernetes-security-webishop?submissionGuid=a82ffed0-0e28-43b2-b23b-635c518995ad

[6] N. Alshuqayran, N. Ali and R. Evans, ”A Systematic Mapping Study in Microservice Architecture,” 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, China, 2016, pp. 44-51, doi: 10.1109/SOCA.2016.15.

[7] Sysdig. ”Prioritize Alerts and Findings with Sysdig Secure” (2023, April). [Online]. Available: https://sysdig.com/blog/prioritize-alerts-and-findings-with-sysdig-secure/

[8] Aquasecurity. ”Kube-bench”. (May 2023). [Online]. Available: GitHub https://github.com/aquasecurity/kube-bench

[9] Aquasecurity. ”Kube-hunter”. (2022). [Online]. Available: GitHub https://github.com/aquasecurity/kube-hunter

[10] Shopify. ”Kubeaudit”. (March 2023). [Online]. Available: GitHub https://github.com/Shopify/kubeaudit

[11] YING, Fei; ZHAO, Shengjie; DENG, Hao. Microservice security framework for IoT by mimic defense mechanism. Sensors, 2022, 22.6: 2418.

[12] Metasploit. ”The world's most used penetration testing framework”. (2023, April). [Online]. Available: https://www.metasploit.com/

[13] MITRE. ”Containers Matrix”. (2023, April). [Online]. Available: https://attack.mitre.org/matrices/enterprise/containers/

Poster: Tight Short-Lived Signatures

Arup Mondal
Ashoka University
Sonipat, Haryana, India
arup.mondal_phd19@ashoka.edu.in

Ruthu Hulikal Rooparaghunath
Vrije Universiteit
Amsterdam, The Netherlands
r.rooparaghunath@student.vu.nl

Debayan Gupta
Ashoka University
Sonipat, Haryana, India
debayan.gupta@ashoka.edu.in

***Abstract*—A Time-lock puzzle (TLP) sends information into the future: a predetermined number of sequential computations must occur (i.e., a predetermined amount of time must pass) to retrieve the information, regardless of parallelization. Buoyed by the excitement around secure decentralized applications and cryptocurrencies, the last decade has witnessed numerous constructions of TLP variants and related applications (e.g., cost-efficient blockchain designs, randomness beacons, e-voting, etc.).**

In this poster, we first extend the notion of TLP by formally defining the "time-lock public key encryption" (TLPKE) scheme. Next, we introduce and construct a "tight short-lived signatures" scheme using our TLPKE. Furthermore, to test the validity of our proposed schemes, we do a proof-of-concept implementation and run detailed simulations.

***Index Terms*—Time-Lock Puzzle, Short-Lived Signatures.**

I. Introduction

A short-lived signature (SLS) provides a verifier with two possibilities: either a generated signature σ on m is correct, or a user has expended a minimum predetermined amount of sequential work (T steps or time) to forge the signature. In other words, the signatures created remain valid for a short period of time T. Formally, we define the unforgeability period as the time starting from when a signer creates a signature for a message using their private signing information (or key) and the `Sign` algorithm. Once the unforgeability period T has elapsed, anyone can compute a forged signature using some public signing information and the `ForgeSign` algorithm.

Recall that any party can compute a forged signature (using `ForgeSign`) after the unforgeability period T has passed. However, there is no guarantee that a party cannot generate a forged signature in *advance*. To ensure this, we use the same model used in [1]. The signature incorporates a random beacon value to ensure it was not created before a specific time T_0. Suppose a verifier observes the signature within $\hat{T}$ units of time after T_0. In this case, they will believe it is a valid signature if $\hat{T} < T$ because it would be impossible to have forged the signature within that time period. Once $\hat{T} \geq T$, the signature is no longer convincing as it may have been constructed through forgery.

Brief Concurrent Work: Recently, Arun et al. [1] studied the variants notion of short-lived cryptographic primitives, i.e., short-lived proofs and signatures. Similar to our work, they make use of sequentially-ordered computations (T-sequential computation) as a means to enforce time delay during which signatures are unforgeable but become forgeable afterward ($(1+c)\cdot T$-sequential computations). In this work, we use the same models as used in [1], however, we define and construct **tight** short-lived signatures, where the forged signatures can be generated in time not much more than sequentially bound T. In other words, tight short-lived signatures ensure that forged signatures can be generated in **exactly** T sequential computations.

TABLE I: Complexity comparison of SLS schemes.

Paper	Setup & Sign	Forge Sign	Verify	Tight
Arun et al. [1]	$\mathsf{poly}(\lambda)$	$O((1+c)\cdot T)$	VDF ([2], [3])	No
Algorithm	$\mathsf{poly}(\lambda)$	$O(T)$	$O(1)$	Yes

Short-lived cryptographic primitives have many real-life use cases; we refer to [1] for a detailed discussion of its applications. *Our main contributions are summarised as follows:*

- First, we extend the time-lock puzzle [4] by formally defining the "time-lock public key encryption" (TLPKE) scheme, and demonstrate a construction using a repeated squaring assumption in a group of unknown order (Sec. III).
- We introduce and construct a "tight short-lived signature" scheme from our TLPKE scheme (Sec. IV).
- We conduct a proof-of-concept implementation study and analyze the performance of our construction (Sec. IV).

II. Technical Preliminaries

Basic Notation: Given a set $\mathcal{X}$, we denote by $x \overset{\$}{\leftarrow} \mathcal{X}$ the process of sampling a value x from the uniform distribution on $\mathcal{X}$. Supp($\mathcal{X}$) denotes the support of the distribution $\mathcal{X}$. We denote by $\lambda \in \mathbb{N}$ the security parameter. A function $\mathtt{negl}$: $\mathbb{N} \rightarrow \mathbb{R}$ is negligible if it is asymptotically smaller than any inverse-polynomial function, namely, for every constant $\epsilon > 0$ there exists an integer N_ϵ and for all $\lambda > N_\epsilon$ such that $\mathtt{negl}(\lambda) \leq \lambda^{-\epsilon}$.

Number Theory: We assume that $N = p \cdot q$ is the product of two large secret and *safe* primes and $p \neq q$. We say that N is a strong composite integer if $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes, where p' and q' are also prime. We say that $\mathbb{Z}_N$ consists of all integers in $[N]$ that are relatively prime to N (i.e., $\mathbb{Z}_N = \{x \in \mathbb{Z}_N : \mathtt{gcd}(x, N) = 1\}$).

Repeated Squaring Assumption: The repeated squaring assumption [4] roughly says that there is no parallel algorithm that can perform T squarings modulo an integer N significantly faster than just doing so sequentially, assuming that N cannot be factored efficiently, or in other words `RSW` assumption implies that factoring is hard. More formally, no adversary can factor an integer $N = p \cdot q$ where p and q are large secrets and "safe" primes (see [2] for details on "safe" primes). Repeated squaring RSW = (Setup, Sample, Eval) is defined below. Moreover, we define a trapdoor evaluation RSW.tdEval (which enables *fast* repeated squaring evaluation), from which we can derive an actual output using trapdoor in $\mathsf{poly}(\lambda)$ time.

- $N \leftarrow$ RSW.Setup(λ) : Output $\mathsf{pp} = (N)$ where $N = p \cdot q$ as the product of two large (λ-bit) randomly chosen secret and safe primes p and q.
- $x \leftarrow$ RSW.Sample(pp) : Sample a random instance x.
- $y \leftarrow$ RSW.Eval(pp, T, x) : Output $y = x^{2^T} \mod N$ by computing the T sequential repeated squaring from x.
- $y \leftarrow$ RSW.tdEval(pp, sp $= \phi(N), x$) : To compute $y = x^{2^T} \mod N$ *efficiently* using the trapdoor as follows:
  - Compute $v = 2^T \mod \phi(N)$.
    Note: $(2^T \mod \phi(N)) \ll 2^T$ for large T
  - Compute $y = x^v \mod N$.
    Note: $x^{2^T} \equiv x^{(2^T \mod \phi(N))} \equiv x^v \pmod N$.

Assumption 1 (T-Repeated Squaring Assumption without Trapdoor [4]). *For every security parameter $\lambda \in \mathbb{N}$, $N \in$ Supp(RSW.Setup(λ)), $x \in$ Supp(RSW.Sample(N)), and a time-bound parameter T, computing the $x^{2^T} \mod N$ without knowledge of a trapdoor or secret parameter* sp *using the* RSW.Eval *algorithm requires T-sequential time for algorithms with $\mathsf{poly}(\log(T), \lambda)$-parallel processors.*

III. Time-Lock Public Key Encryption

The notion of time-sensitive cryptography was introduced by Rivest, Shamir, and Wagner [4] in 1996, in "Time-lock puzzles and timed-release Crypto" (TLP). They presented a construction using repeated squaring in a finite group of unknown order, resulting in an encryption scheme. This scheme allows the holder of a trapdoor to perform "fast" encryption or decryption, while others without the trapdoor can only do so slowly (requiring T sequential computations).

For the purpose of our tight short-lived signature protocol, we require and define a variation of TLP. We follow the definitions given in [4], altered to fit the public key encryption paradigm, rather than symmetric key encryption. This variation, which we refer to as "Time-Lock Public Key Encryption" (TLPKE)[1], can be described as a "public key encryption scheme with sequential and computationally intensive derived private key generation".

Protocol: The formal details of our TLPKE construction from repeated squaring, is TLPKE = (Setup, Eval, Encrypt, Decrypt) specified in Algorithm 1.

Algorithm 1: Time-Lock Public Key Encryption

```
• TLPKE.Setup($\lambda, T$)
    1) Call and generate $N \leftarrow$ RSW.Setup($\lambda$)
    2) Generate an input $x \in \mathbb{Z}_N^* \leftarrow$ RSW.Sample(pp)
    3) Generates a key pair $(pk, sk)$ for a semantically secure public-key encryption
       scheme: PKE = (GenKey, Enc, Dec).
    4) Encrypt the $sk$ as $ek = sk + x^{2^T} \mod N$
    5) Compute $y = x^{2^T} \mod N \in \mathbb{Z}_N^*$ efficiently using the trapdoor evaluation
       RSW.tdEval(pp, sp $= \phi(N), x$).
    6) return pp $= (N, T, x, pk, ek)$.
• TLPKE.Eval(pp)
    1) Compute $y = x^{2^T} \mod N \in \mathbb{Z}_N^*$ using RSW.Eval(pp, $T, x$)
    2) Extract the decryption key $sk = ek - y$
    3) return $(y, sk)$
• TLPKE.Encrypt(pp, $M$)
    1) Encrypt a message $M$ with key $pk$ and a standard encryption Enc, to obtain
       the ciphertext $C_M = \mathrm{Enc}(pk, M \parallel x)$ and return $C_M$.
• TLPKE.Decrypt(pp, $sk, y, C_M$)
    1) Decrypt the message as $M \parallel x = \mathrm{Dec}(sk, C_M)$
    2) Parse $M \parallel x$ and return the message $M$
```

[1]TLP with public key encryption instead of symmetric key encryption

IV. Tight Short-Lived Signature

Syntax and Security Definitions: Here, we recall and modify the definition of short-lived signatures (SLS) from [1] and define our tight SLS as follows:

Definition IV.1 (Tight Short-Lived Signatures). *Let $\lambda \in \mathbb{N}$ be a security parameter and a space of random beacon $\mathcal{R} \geq 2^\lambda$. A short-lived signature* SLS *is a tuple of four probabilistic polynomial time algorithms* (Setup, Sign, ForgeSign, Verify), *as follows:*

- Setup(λ) $\rightarrow$ (pp, sk), *is randomized algorithm that takes a security parameter λ and outputs public parameters* pp *and a secret key sk (the sk can **only** be accessed by the* SLS.Sign *algorithm). The public parameter* pp *contains an input domain $\mathcal{X}$, an output domain $\mathcal{Y}$, and time-bound parameter T.*
- Sign(pp, m, r, sk) $\rightarrow \sigma$, *takes a public parameter* pp, *a secret parameter sp, a message m and a random beacon r, and outputs (in time less than the predefined time bound T) a signature σ.*
- ForgeSign(pp, m, r) $\rightarrow \sigma$, *takes a public parameter* pp, *a message m and a random beacon r, and outputs (in time **exactly** T) a signature σ.*
- Verify(pp, m, r, σ) $\rightarrow$ {accept, reject}, *is a deterministic algorithm takes a public parameter* pp, *a message m and a random beacon r and a signature σ, and outputs* accept *if σ is the correct signature on m and r, otherwise outputs* reject.

A SLS must satisfy the three properties **Correctness** (Definition IV.2), **Existential Unforgeability** (Definition IV.3), and **Indistinguishability** (Definition IV.4) as follows:

Definition IV.2 (Correctness). *A* SLS *is correct (or complete) if for all $\lambda \in \mathbb{N}$, m, and $r \in \mathcal{R}$ it holds that,*

$$Pr\left[\mathtt{Verify}(\mathsf{pp}, m, r, \sigma) = \mathsf{accept} \middle| \begin{array}{l}\mathtt{Setup}(\lambda) \rightarrow (\mathsf{pp}, sk) \\ \mathtt{Sign}(m, r, sk) \rightarrow \sigma\end{array}\right] = 1$$

Definition IV.3 (T-Time Existential Unforgeability). *A* SLS *has T-time existential unforgeability if $\forall\ \lambda, T \in \mathbb{N}$, m and $r \in \mathcal{R}$, and all pairs of PPT algorithms $(\mathcal{A}, \mathcal{A}')$, such an $\mathcal{A}$ (offline) can run in total time $\mathsf{poly}(T, \lambda)$ and in a parallel running time of $\mathcal{A}'$ (online) on at most $\mathsf{poly}(\log T, \lambda)$-processors is less than T, there exists a negligible function* negl *such that,*

$$Pr\left[\begin{array}{l}\mathtt{Sign}(m, r, sk) \rightarrow \sigma \\ \mathtt{Verify}(\mathsf{pp}, m^*, r, \sigma^*) \\ = \mathsf{accept}\end{array} \middle| \begin{array}{l}\mathtt{Setup}(\lambda) \rightarrow (\mathsf{pp}, sk) \\ \mathcal{A}(\mathsf{pp}, \lambda, T) \rightarrow \alpha \\ \mathcal{A}'(\mathsf{pp}, m^*, r, \alpha) \rightarrow \sigma^*\end{array}\right] \leq \mathtt{negl}(\lambda)$$

Definition IV.4 (Indistinguishability)**.** *A* `SLS` *is computationally indistinguishable (and statistically indistinguishable; when taken over the random coins used by each algorithm and randomly generated private parameters) if for all $\lambda \in \mathbb{N}$, m, and $r \in \mathcal{R}$ it holds that,*

$$\left| \begin{array}{l} Pr\left[\mathcal{A}(\mathsf{pp}, m, r, \sigma) = \mathsf{accept} \middle| \begin{array}{l}\mathtt{Setup}(\lambda) \rightarrow (\mathsf{pp}, sk) \\ \mathtt{Sign}(m, r, sk) \rightarrow \sigma\end{array}\right] - \\ Pr\left[\mathcal{A}(\mathsf{pp}, m, r, \sigma) = \mathsf{accept} \middle| \begin{array}{l}\mathtt{Setup}(\lambda) \rightarrow (\mathsf{pp}, sk) \\ \mathtt{ForgeSign}(\mathsf{pp}, m, r) \rightarrow \sigma\end{array}\right] \end{array} \right| \leq \mathtt{negl}(\lambda)$$

Our scheme, formalized in Definition IV.1, presents an efficient generalized framework for short-lived signatures (Algorithm 2) that is compatible with all signature schemes. However, note that while our Indistinguishability definition (Definition IV.4) compares distributions of output, some signature schemes are deterministic (e.g., BLS, RSA signature). In such cases, it is necessary for `Sign` and `Forge` to produce the exact signature (e.g., Schnorr signature) with overwhelming probability.

Protocol Design: The formal construction of tight short-lived signatures `SLS` = (`Setup`, `Sign`, `ForgeSign`, `Verify`) using our TLPKE is specified in Algorithm 2.

Algorithm 2: Tight Short-Lived Signatures from TLPKE

```
• SLS.Setup($\lambda$)
    1) Call and generate $N \leftarrow$ RSW.Setup($\lambda$)
    2) Generate an input $x \in \mathbb{Z}_N^* \leftarrow$ RSW.Sample(pp).
    3) Choose a time bound parameter $T \in T(\lambda)$.
    4) Generates a key pair $(pk, sk) = \Pi_{\mathtt{KeyGen}}(\lambda)$ for a Signature scheme: $\Pi =$ (KeyGen, Sign, Verify).
    5) Compute $y = x^{2^T} \mod N \in \mathbb{Z}_N^*$ efficiently using the trapdoor evaluation RSW.tdEval(pp, sp $= \phi(N), x$).
    6) Encrypts the $sk$ as $ek = sk + x^{2^T} \mod N$
    7) Output public parameter pp $= (N, T, x, pk, ek)$[a] and secret key $sk$ generated by $\Pi$ ($sk$ can only be accessed by the SLS.Sign).
• SLS.Sign($m, r, sk$)
    1) Compute $M = \mathcal{H}(m \parallel r)$.
    2) Compute a signature $\sigma = \Pi_{\mathtt{Sign}}(sk, M)$.
    3) Output a short-lived signature $(\sigma, r)$.
• SLS.ForgeSign(pp, $m, r$)
    1) Compute $M = \mathcal{H}(m \parallel r)$.
    2) Call and extract $(y, sk) =$ TLPKE.Eval(pp).
    3) Compute a forge signature $\sigma = \Pi_{\mathtt{Sign}}(sk, M)$[b].
    4) Output a forge short-lived signature $(\sigma, r)$
• SLS.Verify(pp, $m, r, \sigma$)
    1) Compute $M = \mathcal{H}(m \parallel r)$.
    2) Check that $\Pi_{\mathtt{Verify}}(pk, M, \sigma)$[c].
```

[a]The pp can be generate by calling `TLPKE.Setup`(λ).

[b]Forge signature σ can be computed in time not much more than the sequentiality bound ***exactly*** T even on a parallel computer with poly$(\log T, \lambda)$ processors. Therefore, our SLS is Tight Short-Lived Signature.

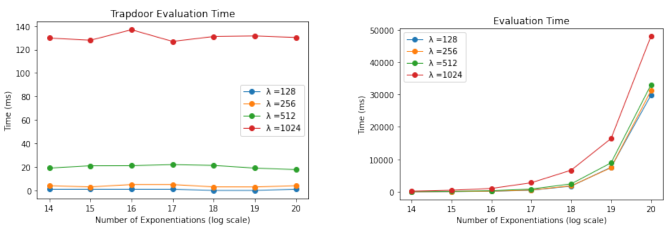
[c]The signature verification algorithm can be computed in $O(1)$ time.

Theorem IV.1 (Tight Short-Lived Signatures)**.** *Assuming that $\mathcal{H}$ is a random oracle,* `RSW` *is the repeated point squaring assumption, and* `TLPKE` *is a time-lock public key encryption scheme (see* Algorithm 1*), it holds that the protocol* `SLS` *(*Algorithm 2*) is a tight short-lived signature scheme.*

Proof Sketch. The correctness of the `SLS` scheme is proven by the correctness of the underlying time-lock public key encryption `TLPKE`. Indistinguishability is trivial as the signing and forgery produce the exact signature using the underlying signature scheme, given that the `TLPKE.Eval` (key extraction) and `TLPKE.Decrypt` (decryption) algorithms of the underlying `TLPKE` are deterministic. The T-Time Existential Unforgeability is a direct result of the sequentiality and security (T-IND-CPA Security) property of the underlying `TLPKE` and modeling $\mathcal{H}$ as a random oracle. □

Experimental Results: We use Python to implement `RSW` primitive (and hence our proposed TLPKE and tight SLS). The experiments are performed using a Windows 11 system with Intel(R) Core(TM) i5-1035G1 CPU @1.00GHz with 8 GB RAM. Note that `RSW` is the underlying required primitive of our repeated squaring-based TLPKE and tight SLS. Hence, in this poster, we do not provide detailed simulation results for TLPKE and tight SLS, focusing instead on profiling the underlying workhorse primitive.

In Figure 1, we show the experimental results for `RSW` evaluation. The `RSW.tdEval` (Figure 1a) run time changes linearly with the security parameter λ (the bit length of N is derived from the bit length of λ). As shown in Figure 1b, the time taken to compute the `RSW.Eval` increases with an increase in the number of exponentiations. Changes in time T yield a great variation in the evaluation time.



(a) Trapdoor Evaluation Time. (b) Evaluation Time.

Fig. 1: Trapdoor Evaluation and T-Sequential Evaluation of the Repeated Squaring `RSW`. j is labelled as "Number of Exponentiations", $T = 2^j$.

V. Future Work

We conclude with an open problem: Arun et al. [1] define *reusable forgeability* property in the context of short-lived proofs (see Sec. 4.1 in [1]), which ensure that one slow computation for a random beacon value (say r) enables efficiently forging a proof for any statement (say x) without performing a full additional slow computation. Furthermore, Arun et al. [1] extend reusable forgeability in the context of short-lived signatures and describe a construction (see Sec. 8.3 in [1]). *In the near future, we hope to construct an efficient* ***tight*** *reusable and forgeable short-lived signature scheme.*

References

[1] A. Arun, J. Bonneau, and J. Clark, "Short-lived zero-knowledge proofs and signatures," *IACR Cryptol. ePrint Arch.*, p. 190, 2022. [Online]. Available: https://eprint.iacr.org/2022/190
[2] K. Pietrzak, "Simple verifiable delay functions," *IACR Cryptol. ePrint Arch.*, 2018. [Online]. Available: https://eprint.iacr.org/2018/627
[3] B. Wesolowski, "Efficient verifiable delay functions," 2018, p. 623. [Online]. Available: https://eprint.iacr.org/2018/623
[4] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," 1996.

Poster: RANDGENER: Distributed Randomness Beacon from Verifiable Delay Function

Arup Mondal
Ashoka University
Sonipat, Haryana, India
arup.mondal_phd19@ashoka.edu.in

Ruthu Hulikal Rooparaghunath
Vrije Universiteit
Amsterdam, The Netherlands
r.rooparaghunath@student.vu.nl

Debayan Gupta
Ashoka University
Sonipat, Haryana, India
debayan.gupta@ashoka.edu.in

***Abstract*—Buoyed by the excitement around secure decentralized applications, the last few decades have seen numerous constructions of distributed randomness beacons (DRB) along with use cases; however, a secure DRB (in many variations) remains an open problem. We further note that it is natural to want some kind of reward for participants who spend time and energy evaluating the randomness beacon value – this is already common in distributed protocols.**

In this work, we present RANDGENER, a *novel* n-party *commit-reveal-recover* (or *collaborative*) DRB protocol with a novel *reward and penalty mechanism* along with a set of realistic guarantees. We design our protocol using trapdoor watermarkable verifiable delay functions in the RSA group setting (without requiring a trusted dealer or distributed key generation).

***Index Terms*—Randomness Beacon, Verifiable Delay Function.**

I. INTRODUCTION

A *randomness beacon* [1] is an ideal functionality that continuously publishes independent random values which no party can predict or manipulate; critically, this value must be efficiently verifiable by anyone. A *Distributed Randomness Beacon* (DRB) protocol allows a set of participants to jointly compute a continuous stream of randomness beacon outputs. A secure DRB protocol should satisfy the following properties, outlined in [2], [3], [4]:

(1) *Liveness/availability*: participants should not be able to prevent the progress of random beacon computation,
(2) *Guaranteed output delivery*: adversaries should not be able to prevent honest participants in the protocol from obtaining a random beacon output,
(3) *Bias-resistance*: no participants should be able to influence future random beacon values to their advantage,
(4) *Public verifiability*: as soon as a random beacon value is generated, it can be verified by anyone independently using only public information, and
(5) *Unpredictability*: participants should not be able to predict the future random beacon values.

We introduce two new desirable properties for DRB protocols: (6) a ***reward mechanism***, which incentivizes participants who invest time and energy in evaluating the randomness beacon value by rewarding their effort, and (7) a ***penalty mechanism***, which discourages inadequate participation, incorrect information or cheating by applying penalties for participants who engage in those actions.

Our n-party distributed randomness beacon protocol, RANDGENER demonstrates a method of claiming "ownership" of a randomness beacon value evaluation in each round of the protocol's execution. This is done by attaching a "watermark" of computing participants to the result of the evaluation in order to reward corresponding participants for their contribution.

Our contributions are summarised as follows:

- We extend watermarkable VDF (wVDF) defined in [5] by formally defining a new type called ***trapdoor*** wVDF. Furthermore, we demonstrate a construction using Wesolowski [5] and Pietrzak's [6] scheme.
- We construct RANDGENER, an efficient n-party *commit-reveal-recover* (or *collaborative*) distributed randomness beacon protocol with a novel ***reward mechanism*** and ***penalty mechanism*** using a trapdoor wVDF. Our protocol does not require any trusted (or expensive) setup and proves that it provides the desired security properties.

Brief Relevant Work: A *commit-reveal* is a classic approach proposed in [1]. First, all participants publish a commitment $y_i = \mathsf{Commit}(x_i)$ to a random value x_i. Next, participants reveal their x_i values, resulting in $R = \mathsf{Combine}(x_1, \dots, x_n)$ for some suitable combination function (such as an exclusive-or or a cryptographic hash).

However, the output can be biased by the last participant to open their commitment (referred to as a *last-revealer attack*), since the last participant, by knowing all other commitments x_i, can compute R early.

A very different approach to constructing DRBs uses time-sensitive cryptography (TSC), specifically using delay functions to prevent manipulation. The simplest example is Unicorn [4], a one-round protocol in which participants directly publish (within a fixed time window) a random input x_i. The result is computed as $R = \mathsf{TSC}(\mathsf{Combine}(x_1, \dots, x_n))$. However, the downside of the Unicorn [4] is that a delay function must be computed for every run of the protocol. Recently, Choi et al. [3] introduced the Bicorn family of DRB protocols, which retain the advantages of Unicorn [4] while enabling efficient computation of the result (with no delay) if all participants act honestly. Yet, as stated in [3], all Bicorn variants come with a fundamental security caveat, i.e., the *last revealer prediction attack*: if participant P_i withholds their x_i value, but all others publish, then participant P_i will be able to simulate efficiently and learn R quickly (*optimistic case*), while honest participants will need to execute the force open and compute the delay function to complete before learning R (*pessimistic case*). Similarly, a coalition of malicious participants can share their x values and privately compute R. Nevertheless, *none* of the existing delay-cryptography-based commit-reveal-recover style DRB protocols provide a reward/penalty mechanism to regulate the behaviour of corrupted participants. In this work, we propose an efficient n-party commit-reveal-recover (or collaborative) DRB protocol with a novel reward and penalty mechanism based on the trapdoor wVDF.

TABLE I: Comparison collaborative DRB schemes.

Paper	Crypto Primit.	Comp. Cost	Comm. Cost	Fault Toler.	Crypto Model			Features								
					Network Model	Setup Assumption	Trusted Setup Req.	Adaptive Adversary	Liveness/Availability	Bias Resistance	Fairness	GOD	Scaleability	Unpredictability	**Reward**	**Penalty**
[4]	Sloth	$O(n)$	$O(1)$	$\frac{(n-1)}{n}$	■	◐	✓	×	✓	✓	✓	✓	×	✓	×	×
[2]	tVDF	$O(n^2)$	$O(1)$	$\frac{n}{2}$	☐	◐	×	✓	✓	✓	✓	✓	✓	×	×	×
[3]	VDF	$O(n^2)$	$O(1)$	$n-1$	☒	◐	×	✓	✓	✓	✓	✓	✓	✓	×	×
Ours	twVDF	$O(n^2)$	$O(1)$	$n-1$	☒	◐	×	✓	✓	✓	✓	✓	✓	✓	✓	✓

☐ denotes the "asynchronous" network model; ☒ denotes the "partial synchronous" network model; ■ denotes the "synchronous" network model. ◐ denotes the "Common Reference String" setup assumption; ✓ denotes provide the property; × denotes does not provide the property. GOD – Guaranteed Output Delivery.

II. Technical Preliminaries

Basic Notation: Given a set $\mathcal{X}$, we denote $x \overset{\$}{\leftarrow} \mathcal{X}$ as the process of sampling a value x from the uniform distribution on $\mathcal{X}$. Supp($\mathcal{X}$) denotes the support of the distribution $\mathcal{X}$.

We denote the security parameter by $\lambda \in \mathbb{N}$. A function $\mathtt{negl}\colon \mathbb{N} \to \mathbb{R}$ is negligible if it is asymptotically smaller than any inverse-polynomial function. Namely, for every constant $\epsilon > 0$ there exists an integer N_ϵ for all $\lambda > N_\epsilon$ such that $\mathtt{negl}(\lambda) \le \lambda^{-\epsilon}$.

Number Theory: We assume that $N = p \cdot q$ is the product of two large secret and *safe* primes and $p \neq q$. We say that N is a strong composite integer if $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes, where p' and q' are also prime. We say that $\mathbb{Z}_N$ consists of all integers in $[N]$ that are relatively prime to N (i.e., $\mathbb{Z}_N = \{x \in \mathbb{Z}_N : \mathtt{gcd}(x, N) = 1\}$).

Repeated Squaring Assumption: The repeated squaring assumption [6] roughly states that there is no parallel algorithm that can perform T squarings modulo an integer N significantly faster than just doing so sequentially, assuming that N cannot be factored efficiently, or in other words `RSW` assumption implies that factoring is hard. More formally, no adversary can factor an integer $N = p \cdot q$ where p and q are large secret and "safe" primes [6]. A repeated squaring `RSW = (Setup, Sample, Eval)` is defined below. Moreover, we define a trapdoor evaluation `RSW.tdEval` (to enable *fast* repeated squaring evaluation), from which we can derive an actual output using trapdoor in $\mathsf{poly}(\lambda)$ time.

- $N \leftarrow \mathtt{RSW.Setup}(\lambda)$: Output $\mathsf{pp} = (N)$ where $N = p \cdot q$ as the product of two large (λ-bit) randomly chosen secret and safe primes p and q.
- $x \leftarrow \mathtt{RSW.Sample}(\mathsf{pp})$: Sample a random instance x.
- $y \leftarrow \mathtt{RSW.Eval}(\mathsf{pp}, T, x)$: Output $y = x^{2^T} \mod N$ by computing the T sequential repeated squaring from x.
- $y \leftarrow \mathtt{RSW.tdEval}(\mathsf{pp}, \mathsf{sp} = \phi(N), x)$: To compute $y = x^{2^T} \mod N$ *efficiently* using the trapdoor as follows: ***(i)*** Compute $v = 2^T \mod \phi(N)$. **Note:** $(2^T \mod \phi(N)) \ll 2^T$ for large T. ***(ii)*** Compute $y = x^v \mod N$. **Note:** $x^{2^T} \equiv x^{(2^T \mod \phi(N))} \equiv x^v \pmod N$.

III. Verifiable Delay Function

A *verifiable delay function* (VDF), introduced by Boneh et al. [7], is a special type of delay function f characterized by a time-bound parameter T and the following three properties: ***(i)*** *T-sequential function*: The function f can be evaluated in sequential time T, but it should not be possible to evaluate f significantly faster than T even with parallel processing. ***(ii)*** *Unique output*: The function f produces a unique output, which is efficiently and publicly ***(iii)*** *Verifiable* (in time that is essentially independent of T) - meaning that the function f should produce a proof π which convinces a verifier that the function output has been correctly computed.

Wesolowski [5] first describes a trapdoor VDF (tVDF) as a modified and extended version of traditional VDFs [7] such that the `Setup` algorithm, in addition to the public parameters pp, outputs a trapdoor or secret parameter sp to the party invoking the `Setup` algorithm. This parameter sp is kept secret by the invoker, whereas pp is published. Furthermore, using the trapdoor evaluation `tdEval` and trapdoor proof generation `tdProve` (by enabling *fast* computations), the secret parameter-holding participants can derive an actual output and the proof of correctness in $\mathsf{poly}(\lambda)$ time. Parties without knowledge of the trapdoor, as in the traditional VDF case, can still compute the output and proof of correctness by executing `Eval` and `Prove`. However, it requires T-sequential steps to do so. For the purpose of our distributed random beacon protocol, we require and define a trapdoor watermarkable VDF (twVDF). In this case, we use the same trapdoor evaluation `tdEval`, but we generate a watermarked proof of correctness using `tdProve` by embedding a watermark of the evaluator.

In Algorithm 1, we provide details for the formal construction of watermarkable verifiable delay function `VDF` using Wesolowski [5] and Pietrzak's [6] scheme, consisting of algorithms (`Setup`, `Sample`, `Eval`, `Prove`, `Verify`) with a trapdoor watermarkable VDF evaluation `tdEval` and proof generation `tdProve`.

Algorithm 1: Trapdoor Watermarkable VDF using Wesolowski and Pietrzak

```
• VDF.Setup($\lambda$)
    1) Call and generate $N \leftarrow \mathtt{RSW.Setup}(\lambda)$
    2) A cryptographically secure $\lambda$-bit hash function $\mathcal{H}_{\mathsf{prime}}$ or $\mathcal{H}_{\mathsf{random}}$.
    3) Generate a time-bound parameter $T$.
    4) return $\mathsf{pp} = (N, T, \mathcal{H})$.
  • VDF.Sample($\mathsf{pp}$)
    1) Generate an input $x \in \mathbb{Z}_N^* \leftarrow \mathtt{RSW.Sample}(\mathsf{pp})$
  • VDF.Eval($\mathsf{pp}, x$)
    1) Compute $y = x^{2^T} \mod N \in \mathbb{Z}_N^*$ using $\mathtt{RSW.Eval}(\mathsf{pp}, T, x)$
    2) Generate an advice string $\alpha$.
    3) return $(y, \alpha)$.
  • VDF.tdEval($\mathsf{pp}, x$)
    1) Compute group order $\phi(N) = (p-1) \cdot (q-1)$ using trapdoor $(p, q)$.
    2) Compute $y = x^{2^T} \mod N$ using RSW.tdEval($\mathsf{pp}, \mathsf{sp} = \phi(N), x$).
    3) Generate an advice string $\alpha$.
    4) return $(y, \alpha)$.
.................... Using Wesolowski's [5] Scheme ....................
  • VDF.Prove($\mathsf{pp}, x, \mu, y, \alpha, T$)
    1) Generate a prime $l = \mathcal{H}_{\mathsf{prime}}(x \parallel y \parallel \mu)$ [a]
    2) Compute the proof $\pi_\mu = x^{\lfloor 2^T / l \rfloor} \pmod N$ [b] and return $\pi_\mu$.
  • VDF.tdProve($\mathsf{pp}, x, \mu, y, \alpha, T$)
    1) Generate a prime $l = \mathcal{H}_{\mathsf{prime}}(x \parallel y \parallel \mu)$
    2) Compute group order $\phi(N) = (p-1) \cdot (q-1)$ using trapdoor $(p, q)$.
    3) Compute proof $\pi_\mu = x^{(\lfloor 2^T / l \rfloor \mod \phi(N))} \pmod N$ and return $\pi_\mu$.
  • VDF.Verify($\mathsf{pp}, x, \mu, y, \pi_\mu, T$)
    1) Generate a prime $l = \mathcal{H}_{\mathsf{prime}}(x \parallel y \parallel \mu)$
    2) $r = 2^T \mod l$
    3) return accept if $(\pi_\mu^l \cdot x^r) \mod N = y$, otherwise reject
...................... Using Pietrzak's [6] Scheme ......................
  • VDF.Prove($\mathsf{pp}, x, \mu, y, \alpha, T$)
    1) Compute $u = x^{2^{T/2}} \mod N$
    2) Generate a random $r = \mathcal{H}_{\mathsf{random}}(x \parallel T/2 \parallel y \parallel u \parallel \mu)$ [c]
    3) Compute $x = x^r \cdot u \mod N$ and $y = u^r \cdot y \mod N$
    4) Proof $\pi_\mu = u \cup$ VDF.Prove($\mathsf{pp}, x, \mu, y, \alpha, T/2$) and return $\pi_\mu$.
  • VDF.tdProve($\mathsf{pp}, x, \mu, y, \alpha, T$)
    1) Compute group order $\phi(N) = (p-1) \cdot (q-1)$ using trapdoor $(p, q)$.
    2) Compute $u = x^{(2^{T/2} \mod \phi(N))} \mod N$
    3) Generate a random $r = \mathcal{H}_{\mathsf{random}}(x \parallel T/2 \parallel y \parallel u \parallel \mu)$
    4) Compute $x = x^{(r \mod \phi(N))} \cdot u \mod N$ and $y = u^{(r \mod \phi(N))} \cdot y \mod N$
    5) Proof $\pi_\mu = u \cup$ VDF.tdProve($\mathsf{pp}, x, \mu, y, \alpha, T/2$) and return $\pi_\mu$.
  • VDF.Verify($\mathsf{pp}, x, \mu, y, \pi_\mu, T$)
    1) Generate a random $r = \mathcal{H}_{\mathsf{random}}(x \parallel T/2 \parallel y \parallel u \parallel \mu)$
    2) Compute $x = x^r \cdot u \mod N$ and $y = u^r \cdot y \mod N$
    3) Call VDF.Verify($\mathsf{pp}, x, \mu, y, \pi_\mu, T/2$)
    4) return accept if $T = 1$ check $y = x^2 \mod N$, otherwise reject.
```

[a] Sampled uniformly from $\mathsf{Prime}(\lambda)$

[b] μ is an evaluator's watermark

[c] Sampled uniformly from $\{1, 2, \ldots, 2^\lambda\}$

IV. RandGener Protocol Design

In this section, we present our RandGener protocol, a n-party distributed randomness beacon protocol DRB = (Setup, VerifySetup, Gen). The construction details are in Algorithm 2 using our trapdoor watermarkable VDF.

Algorithm 2: RandGener: Distributed Randomness Beacon Protocol

Input: A globally agreed security parameter λ, a set of participants $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$, a set of public parameters $\mathcal{PP} = \{\mathsf{pp}_1, \mathsf{pp}_2, \ldots, \mathsf{pp}_n\}$, a time-bound parameter T, an initial random beacon value R_0 (it becomes available to all parties running the protocol after the setup is completed at approximately the same time), and two cryptographically secure λ-bit hash functions: (i) $\mathcal{H}_{\mathsf{randToinput}}$ – mapping a random value to the input space of the VDF, and (ii) $\mathcal{H}_{\mathsf{inputTorand}}$ – mapping a VDF output to a random value.
Output: The randomness beacon value $R_1, R_2, \ldots, R_\infty$ for that round of the protocol.

```
  • DRB.Setup($\lambda$)
    1) $\forall i$ $P_i \in \mathcal{P}$ locally generate a public parameter $\mathsf{pp}_i = (N_i, T, \mathcal{H}) =$ VDF.Setup($\lambda$).
    2) $\forall i$ $P_i \in \mathcal{P}$ run the zero-knowledge protocol for proving that a known $N_i$ is
       the product of two safe primes and the protocol "proving the knowledge of a
       discrete logarithm that lies in a given range" to show that the prime factors
       $p_i$ and $q_i$ are $\lambda$-bits each. Let $\pi_{N_i}$ denote the resulting proof obtained by
       running both protocols non-interactively using the Fiat-Shamir heuristic.
    3) Broadcast ($\mathcal{PP} = \{\mathsf{pp}_1, \ldots, \mathsf{pp}_n\}$, $\Pi = \{\pi_{N_1}, \ldots, \pi_{N_n}\}$).
  • DRB.VerifySetup($\mathcal{PP}, \Pi$)
    1) For each public parameter $\mathsf{pp}_i \in \mathcal{PP}$ and a corresponding proof $\pi_{N_i} \in \Pi$,
       return accept if the validity of $\mathsf{pp}_i$ can be successfully checked by using
       the verification procedures corresponding to the proof techniques used in
       DRB.Setup algorithm as specified in [2], otherwise return reject.
  • DRB.Gen($\mathcal{PP}, T, R_0$)
    1) Set $r \leftarrow 1$.
    ...................... Commit ...................... deadline $T_0$
    2) Compute $x_r \leftarrow \mathcal{H}_{\mathsf{randToinput}}(R_{r-1})$.
    3) Generate a random input $x'_{r,i}$
    4) Compute and publish $x_{r,i} \leftarrow \mathcal{H}(x'_{r,i} || x'_r)$                ▷ Broadcast
    ...................... Reveal ...................... deadline $T_1$
    5) For each participant $P_i \in \mathcal{P}$ in parallel
       a) Compute $(y_{r,i}, \alpha_{r,i}) \leftarrow$ VDF.tdEval($\mathsf{pp}_i, x_{r,i}, T$).
       b) Compute $\pi_{r,i} \leftarrow$ VDF.tdProve($\mathsf{pp}_i, x_{r,i}, \mu_i, y_{r,i}, \alpha_{r,i}, T$).
       c) Publish $(y_{r,i}, \pi_{r,i})$.                                   ▷ Broadcast
    ............................ Finalize ............................
    6) For each participant $P_i \in \mathcal{P}$, verify $(x_r, y_{r,i}, \pi_{r,i})$
       a) If VDF.Verify($\mathsf{pp}_i, x_{r,i}, y_{r,i}, \pi_{r,i}, T$) = reject or $x_{r,i}$ was not
          published by $T_1$, then remove participant $P_i$ and add $\tilde{\mathcal{P}} \leftarrow \tilde{\mathcal{P}} \cup P_i$.
    7) For all $P_i \in \mathcal{P}$, If VDF.Verify($\mathsf{pp}_i, x_{r,i}, y_{r,i}, \pi_{r,i}, T$) = accept
       a) Compute $y_r = \prod_{P_i \in \mathcal{P}} y_{r,i}$                         ▷ Optimistic case
       b) Optionally, a proof $\pi_{y_r}$ can be compute to enable verification of $y_r$.
    ............................ Recover ............................
    8) For each participant $P_j \in \tilde{\mathcal{P}}$ in parallel                  ▷ Pessimistic case
       a) Compute $(y_{r,j}, \alpha_{r,j}) \leftarrow$ VDF.Eval($\mathsf{pp}_j, x_{r,j}, T$).
       b) Compute $\pi_{r,j} \leftarrow$ VDF.Prove($\mathsf{pp}_j, x_{r,j}, \mu_r, y_{r,j}, \alpha_{r,j}, T$).
       c) Compute $y_r = \prod_{P_i \in \mathcal{P}} y_{r,i} \cdot \prod_{P_j \in \tilde{\mathcal{P}}} y_{r,j}$
       d) Optionally, a proof $\pi_{y_r}$ can be computed to enable verification of $y_r$.
    ..................................................................
    9) Output the $r$-th round's randomness beacon $R_r \leftarrow \mathcal{H}_{\mathsf{inputTorand}}(y_r)$
   10) Reward the participants computing the $r$-th round's randomness beacon
       value $\mathcal{P} \setminus \tilde{\mathcal{P}}$ and apply a Penalty to participants $\tilde{\mathcal{P}}$.
   11) Set $r \leftarrow r + 1$.
   12) Repeat from step 2 to step 11 – to generate the next round's randomness.
```

Theorem IV.1. *Assuming that $\mathcal{H}_{\mathsf{randToinput}}$ and $\mathcal{H}_{\mathsf{inputTorand}}$ is the random oracle and* VDF *is a trapdoor watermarkable VDF, then it holds that* Algorithm 2 *is a DRB scheme.*

The proof of Theorem IV.1 is deferred to the full version.

V. Future Work

Existing collaborative DRB protocols experience challenges in inefficient communication complexity, which limits their scalability.

In the near future, we hope to construct a complexity-efficient collaborative DRB protocol.

References

[1] M. Blum, "Coin flipping by telephone a protocol for solving impossible problems," *SIGACT News*, vol. 15, no. 1, pp. 23–27, 1983. [Online]. Available: https://doi.org/10.1145/1008908.1008911

[2] P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. R. Weippl, "Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness," 2020, p. 942. [Online]. Available: https://eprint.iacr.org/2020/942

[3] K. Choi, A. Arun, N. Tyagi, and J. Bonneau, "Bicorn: An optimistically efficient distributed randomness beacon," *IACR Cryptol. ePrint Arch.*, p. 221, 2023. [Online]. Available: https://eprint.iacr.org/2023/221

[4] A. K. Lenstra and B. Wesolowski, "A random zoo: sloth, unicorn, and trx," *IACR Cryptol. ePrint Arch.*, p. 366, 2015. [Online]. Available: http://eprint.iacr.org/2015/366

[5] B. Wesolowski, "Efficient verifiable delay functions," 2018, p. 623. [Online]. Available: https://eprint.iacr.org/2018/623

[6] K. Pietrzak, "Simple verifiable delay functions," *IACR Cryptol. ePrint Arch.*, 2018. [Online]. Available: https://eprint.iacr.org/2018/627

[7] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 601, 2018. [Online]. Available: https://eprint.iacr.org/2018/601

Poster: MaSTer: (Practically) Maliciously Secure Truncation for Replicated Secret Sharing

Martin Zbudila Erik Pohle Aysajan Abidin Bart Preneel
imec-COSIC, KU Leuven
Belgium
firstname.lastname@esat.kuleuven.be

***Abstract*—Secure multi-party computation (MPC) in a three-party honest majority scenario is currently the state-of-the-art for running machine learning algorithms in a privacy-preserving manner. For efficiency reasons, fixed-point arithmetic is widely used to approximate computation over decimal numbers. In this poster we present a work-in-progress efficient three-party maliciously secure truncation protocol without pre-processing, thereby improving on the current state-of-the-art in MPC over rings using replicated secret sharing (RSS).**

1. Introduction

Machine learning (ML) and AI dominate the world of data processing, and the need to preserve the privacy of both the model owners' and users' data becomes an important objective. Numerous works on privacy-preserving machine learning use advanced cryptographic techniques such as secure multi-party computation (MPC) [2]–[4] to enable confidential training and inference of various machine learning models. In MPC, the state-of-the-art is computation in the three-party setting with honest majority.

A key component in evaluating ML algorithms over MPC is the efficient computation on secret-shared decimal numbers. Since true floating point arithmetic is expensive, an approximation, namely fixed-point arithmetic, is employed. Here, the decimal number is encoded with fixed precision as an integer where the integer size is chosen to accommodate the expected range. Multiplication of two fixed-point integers doubles the length of the fractional part and thus requires efficient truncation afterwards to retain the original precision. For efficient secret sharing in MPC, the integer is represented in a ring. SecureML [3] first proposed a technique allowing truncation by local operations on the shares in a two-party setting in semi-honest security. The current state-of-the-art three-party maliciously secure truncation was proposed by ABY3 [2], and adopted by follow up works such as Falcon [4]. In the online phase, this truncation requires only one round of communication and only one ring element sent per party. However, the pre-processing phase involves a costly evaluation of subtraction circuits, resulting in lower combined throughput.

In this poster, we report on a work-in-progress protocol that computes the truncation of fixed-point numbers without pre-processing securely in the presence of one malicious adversary in the three-party setting. Our protocol achieves an overall low cost of two rounds of communication and one ring element sent per party.

Idea. Our protocol is based on the two-party truncation from SecureML [3]. We first run a semi-honest truncation in the first round and repeat the semi-honest truncation a second time but with a different subset of parties. Finally, the output of the second round is used to verify that no malicious behaviour tampered the result obtained in the first round.

In the following, we briefly present necessary preliminaries in Sect. 2 and describe our protocol for truncation in Sect. 3.

2. Preliminaries

We begin by presenting the notation used throughout this poster, and discuss related work.

2.1. Notation

Given a fixed-point encoded secret $x \in (-2^{\ell_x}, 2^{\ell_x})$, we define z to be the encoding of

x in the ring $\mathbb{Z}_{2^\ell}$, such that $z = x$ if $x \geq 0$ and $z = 2^\ell - x$ if $x < 0$. Here, ℓ_x denotes the length of the input x, such that $\ell_x < \ell - 1$. We use the notation of $[\![z]\!]$ to denote a 2-out-of-3 replicated secret sharing (RSS), where $[\![z]\!] = (s_1, s_2, s_3)$, such that $s_1 + s_2 + s_3 = z$. We use $[\![z]\!]_i = (s_i, s_{i+1})$ for $1 \leq i \leq 3$ to denote the shares of individual parties. Further, we use $\langle z \rangle$ to denote 2-out-of-2 additive sharing of z, i.e., $\langle z \rangle_1 + \langle z \rangle_2 = z$ and $\langle z \rangle_i$ denotes the share of party i. If z is the result of a fixed-point multiplication, we denote the truncation of z by ℓ_D bits as $\lfloor z \rfloor$. We use the notation $r \xleftarrow{\$_{i,j}} \mathbb{Z}_{2^\ell}$ to denote $r \in \mathbb{Z}_{2^\ell}$ sampled uniformly at random from shared randomness between parties i and j.

2.2. Related work

In SecureML [3], the authors observe that the following local operations on a two-party additive sharing of z amount to a truncation with an error on the least significant bit (LSB) with a large probability. If party P_1 computes $\langle \lfloor z \rfloor \rangle_1 = \lfloor \langle z \rangle_1 \rfloor$ and party P_2 computes $\langle \lfloor z \rfloor \rangle_2 = 2^\ell - \lfloor 2^\ell - \langle z \rangle_2 \rfloor$ locally, where $\lfloor \langle \cdot \rangle \rfloor$ denotes a truncation, i.e., right shift, on the share value, then the result is a sharing of $\lfloor z \rfloor \pm 1$ with probability $1 - 2^{\ell_x + 1 - \ell}$ (see [3, Theorem 1]).

ABY3 [2] introduces two truncation methods for three-party RSS schemes. The authors note that the local truncation of SecureML fails when naively extended to three parties. Nevertheless, assuming semi-honest security, two parties can transform a RSS $[\![z]\!]$ to a 2-out-of-2 sharing $\langle z \rangle$,perform the SecureML truncation with local operations and reshare the result to obtain shares of $[\![\lfloor z \rfloor]\!]$. However, the above approach is not secure in the presence of a malicious adversary. ABY3 therefore proposes another technique to achieve a maliciously secure truncation. The algorithm assumes a pre-processed pair of shares of random $[\![r]\!]$ and $[\![\lfloor r \rfloor]\!]$, from which the parties are able to compute $[\![\lfloor z \rfloor]\!] = [\![\lfloor r \rfloor]\!] + \lfloor (z - r) \rfloor$. However, the pre-processing is heavy in communication, requiring $\mathcal{O}(\ell)$ rounds.

3. MaSTer: Maliciously secure truncation for replicated secret sharing

In Sect. 3.1 and Fig. 1, we describe the idea behind our maliciously secure truncation protocol without the need for an offline phase. In Sect. 3.2 and 3.3 we briefly describe our results regarding the truncation error bounds and give a sketch proof of security.

3.1. Protocol

Our protocol combines the SecureML truncation with the extension to a three-party scenario in semi-honest setting as suggested by ABY3. The protocol (see Fig. 1) consists of two rounds. The first round is analogous to the semi-honest truncation with resharing afterwards and results in *unchecked* shares of the truncated multiplication result in RSS. In our case, the only way a malicious party can influence the result is by controlling P_1 in the first round and sending $s'_2 + \delta$, where δ is an additive error introduced by the malicious party. In order to detect this potential malicious behaviour in the first round, we introduce a second round where parties P_2 and P_3 compute the two-party truncation, such that P_1 does not contribute. Therefore, P_2 and P_3 now hold two independently created sharings of the truncated value, allowing them to compute the difference and hence verifying the correctness of the value sent by P_1, up to an error of ± 1. We will refer to the verification as a γ-check.

3.2. Error Analysis

In Theorem 1 we formalise the probability of a successful execution of our protocol, verifying the correctness and consistency of the γ-check. Upon closer analysis of the error potentially occurring in each execution of the SecureML truncation, we can conclude that despite the execution of two independent truncations, the difference between the two results can be bounded by ± 1. This is because the two truncations are performed on a sharing with the same randomness.

Theorem 1 (Correctness of consistency check)**.** *In an honest execution of Π_{Trunc} (see Fig. 1), the consistency check holds with probability* $\Pr\left[\sum_{i=1}^{3} \gamma_i \in \{0, \pm 1\}\right] \geq 1 - 2^{\ell_x + 2 - \ell}$.

It is important to note that in the case of triggering the failure condition of the SecureML probabilistic truncation, the obtained result will include a very large error instead of the small error of ± 1. Such error can result in large γ value causing an abort with no misbehaviour of a malicious party. However, based on Theorem 1 we

Truncation protocol Π_{Trunc}

P_1		P_2		P_3
$[\![z]\!]_1 = (s_1, s_2)$		$[\![z]\!]_2 = (s_2, s_3)$		$[\![z]\!]_3 = (s_3, s_1)$
$s'_1 := r' \xleftarrow{\$_{1,3}} \mathbb{Z}_{2^\ell}$				$s'_1 := r' \xleftarrow{\$_{1,3}} \mathbb{Z}_{2^\ell}$
$s'_2 := \mathsf{rshift}(s_1 + s_2) - r'$	$\xrightarrow{s'_2}$	$s'_3 := 2^\ell - \mathsf{rshift}(2^\ell - s_3)$		$s'_3 := 2^\ell - \mathsf{rshift}(2^\ell - s_3)$
		$s''_3 := r'' \xleftarrow{\$_{2,3}} \mathbb{Z}_{2^\ell}$		$s''_3 := r'' \xleftarrow{\$_{2,3}} \mathbb{Z}_{2^\ell}$
		$s''_2 := 2^\ell - \mathsf{rshift}(2^\ell - s_2)$		$s''_1 := \mathsf{rshift}(s_3 + s_1) - r''$
		$\gamma_2 = s'_2 - s''_2$	$\xrightarrow{\gamma_2}$	$\gamma_1 = s'_1 - s''_1$
		$\gamma_3 = s'_3 - s''_3$	$\xleftarrow{\gamma_1}$	$\gamma_3 = s'_3 - s''_3$
		Check $\lvert \sum \gamma_i \rvert \leq 1$		Check $\lvert \sum \gamma_i \rvert \leq 1$
Output $[\![\lfloor z \rfloor]\!]_1 := (s'_1, s'_2)$		Output $[\![\lfloor z \rfloor]\!]_2 := (s'_2, s'_3)$		Output $[\![\lfloor z \rfloor]\!]_3 := (s'_3, s'_1)$

Figure 1. The truncation protocol Π_{Trunc}.

can conclude that our protocol aborts incorrectly with negligible probability for large ℓ.

3.3. Security Proof Sketch

We prove that Π_{trunc} securely realises $\mathcal{F}_{\mathsf{trunc}}$ against a static malicious adversary who corrupts a single party.

Proof. (Sketch.) We give a simulator $\mathcal{S}$ that simulates the honest parties towards an adversary in an ideal execution with access to $\mathcal{F}_{\mathsf{trunc}}$. The goal is to show that this interaction is indistinguishable for the environment from the interaction between the real adversary and the real protocol. Notably, $\mathcal{S}$ does not know the inputs of the honest parties it simulates. However, in the input phase, the adversary secret-shares its inputs with the honest parties/simulator and thus $\mathcal{S}$ always knows the shares of the corrupted party and can track them throughout the whole computation. If P_1 is corrupted, $\mathcal{S}$ receives s'_2 from $\mathcal{A}$ and since $\mathcal{S}$ knows s_2, can reproduce the computation and fail accordingly. If P_2 is corrupted, $\mathcal{S}$ sends a random value in place of s'_2 and adapts γ_1 s.t. the check will still hold. If P_3 is corrupted, $\mathcal{S}$ reproduces γ_1 and γ_3 and sends a γ_2 that is consistent. In any case, $\mathcal{S}$ uses the ideal truncation functionality $\mathcal{F}_{\mathsf{trunc}}$ to prepare the output shares. □

4. Experiments

We run our experiments in the MP-SPDZ [1] framework. Our setup consists of three servers with 16GB RAM each and the network configuration: (i) LAN: 10Gbit throughput, 0.1ms round trip time (RTT), (ii) WAN: 50Mbps throughput, 40ms RTT. We report the performance results in Table 1.

TABLE 1. INFERENCE OF ONE SAMPLE, TOTALLING 17100 TRUNCATIONS.

	Offline		Online		Throughput		Data
	LAN	WAN	LAN	WAN	LAN	WAN	sent
ABY3	0.12	9.35	0.026	2.41	410	5	25.76
MaSTer			0.033	3.657	1818	16	0.407

For a single query we improve the total running time by a factor of three. The evaluation time is displayed in seconds, data sent in MB and the throughput denotes the number of inferences per minute.

References

[1] Marcel Keller. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[2] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 35–52, 2018.

[3] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38, 2017.

[4] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. F: Honest-majority maliciously secure framework for private deep learning. *Proceedings on Privacy Enhancing Technologies*, 2021(1):188–208, 2021.