ACM DIGITAL LIBRARY · Association for Computing Machinery · acm open

RESEARCH-ARTICLE

Privacy-Preserving Runtime Verification

THOMAS A HENZINGER, Institute of Science and Technology Austria (ISTA), Klosterneuburg, Lower Austria, Austria

MAHYAR KARIMI, Institute of Science and Technology Austria (ISTA), Klosterneuburg, Lower Austria, Austria

K S THEJASWINI, Institute of Science and Technology Austria (ISTA), Klosterneuburg, Lower Austria, Austria

Privacy-Preserving Runtime Verification

Thomas A. Henzinger
tah@ista.ac.at
ISTA
Klosterneuberg, Austria

Mahyar Karimi
mahyar.karimi@ista.ac.at
ISTA
Klosterneuberg, Austria

K. S. Thejaswini
thejaswini.k.s@ista.ac.at
ISTA
Klosterneuberg, Austria

Abstract

Runtime verification offers scalable solutions to improve the safety and reliability of systems. However, systems that require verification or monitoring by a third party to ensure compliance with a specification might contain sensitive information, causing privacy concerns when usual runtime verification approaches are used. Privacy is compromised if protected information about the system, or sensitive data that is processed by the system, is revealed. In addition, revealing the specification being monitored may undermine the essence of third-party verification.

In this work, we propose two novel protocols for the privacy-preserving runtime verification of systems against formal sequential specifications. In our first protocol, the monitor verifies whether the system satisfies the specification without learning anything else, though both parties are aware of the specification. Our second protocol ensures that the system remains oblivious to the monitored specification, while the monitor learns only whether the system satisfies the specification and nothing more. Our protocols adapt and improve existing techniques used in cryptography, and more specifically, multi-party computation.

The sequential specification defines the observation step of the monitor, whose granularity depends on the situation (e.g., banks may be monitored on a daily basis). Our protocols exchange a single message per observation step, after an initialisation phase. This design minimises communication overhead, enabling relatively lightweight privacy-preserving monitoring. We implement our approach for monitoring specifications described by register automata and evaluate it experimentally.

CCS Concepts

• **Theory of computation** → **Logic and verification**; *Interactive computation*; **Cryptographic protocols**; **Formalisms**.

Keywords

Privacy-preserving verification; Runtime verification; Monitoring ; Reactive functionalities

1 Introduction

Recent advances have demonstrated that verification can be performed in a privacy-preserving manner spanning a variety of domains, including SAT solving [33], verifying resolution proofs [32], matching strings against regular expressions [34], and also model-checking specifications described by CTL formulas [24]. However, privacy-preserving verification often introduces significant computational overhead. This poses scalability challenges, particularly in real-world applications where the underlying systems involve an enormous number of states. In contrast, runtime verification [4, 30] focuses not on verifying the entire system but rather on monitoring specific outputs produced by the system during execution. This approach inherently involves smaller-scale data exchanges, making it a suitable candidate for privacy-preserving methods.

Privacy and monitoring seem at odds. Monitoring typically involves verifying whether the execution trace of a system satisfies a given specification, which seemingly requires access to the system trace. Privacy, on the other hand, demands that protected information about the system remain undisclosed. Balancing these conflicting objectives presents a significant challenge, particularly in contexts where monitoring is essential but the underlying data is sensitive. This challenge arises in many real-world scenarios; a bank undergoing an audit may need to share data to demonstrate compliance, a hospital may need to report statistics to ensure adherence to healthcare standards, a self-driving car or metro system may be monitored for its operational safety. In each case, sharing protected IP or client data with a third-party monitor raises concerns about privacy. One possible solution is to allow the system to self-monitor by embedding the specification into the system and generating its own verification results. However, this approach lacks credibility in settings where third-party validation is crucial. For instance, if hospitals were entirely self-certified, the certificates would lack the impartiality needed to inspire public confidence. This necessitates a solution that allows third-party monitors to continuously and repeatedly verify whether a system complies with a given specification, all while obscuring the internals of the system and its data.

Hiding the system and its data is just a first step, but our need for privacy might not stop there. Sometimes, one might even need to hide the specification that a system is being monitored against. For motivation, we can again consider the domain of healthcare, where the runtime verification of traces satisfying specifications expressed in temporal logic has already been studied [23]. For instance, in the National Health Services (NHS) in England, hospital funding is tied to performance metrics, optimisation of which led many hospitals to restructure their operations [17, 36]. Unfortunately,

---

such restructurings resulted in extreme cases like the events at Mid-Staffordshire NHS Foundation Trust, where financial targets led to patient neglect [22]. This exemplifies Goodhart's Law: *"When a measure becomes a target, it ceases to be a good measure,"* or, as an ex-NHS manager put it, *"hitting the target but missing the point."* Indeed any measure has a potential to become a target when gamifications are employed to achieve that target. To avoid such distortions, it is crucial to also develop protocols for monitoring that keep both the system/data and specification private, ensuring accurate and unbiased evaluation.

Cryptography. We use tools developed in cryptography to accomplish our privacy-preserving algorithms in the settings described above. Our privacy-preserving monitoring algorithms rely on techniques from multi-party computation (MPC), studied since the 1980s [42]. MPC allows two or more parties, who do not trust each other, to collaboratively compute a function on their private inputs without revealing these inputs to each other. Typically, MPC focuses on a "one-shot" setting, where the parties compute one or more outputs based on their inputs, but do so exactly once. We use a specific variant of MPC called private function evaluation (PFE), where one party owns a private function (a "secret circuit") along with a part of the input, while the other party holds the rest of the input to that function. The goal is to design a protocol that allows the function to be evaluated using the inputs of both parties securely: one or both parties can learn the result of the function on the input, but nothing else is revealed. Essentially, PFE enables computation as if the function owner shared their circuit and the other shared their input, without actually doing so.

Runtime verification, however, requires not a one-shot but a repeated evaluation process, whose granularity—called (*observation*) *round*—depends on the application. The performance of banks may be observed daily, hospitals monthly, autopilots and metro systems every second. Although most existing PFE protocols are designed for single-use computations, a few have been modified for efficient repeated use [7, 31, 37]; however, these usually do not account for situations where the computation needs to be repeated many times, nor for maintaining internal states that are kept secret from all parties. Such *reactive functionalities* are important for monitoring systems, where internal states of the system (e.g., accumulated data or ongoing logs) and the monitor (for sequential specifications) need to remain hidden even across repeated interactions. Protocols that can be adapted to reactive functionalities using standard cryptographic techniques like secret sharing require at least as many message exchanges per round as Oblivious Transfer (cryptographic protocols in which a receiver is to obtain one of many messages from a sender without revealing their choice), which exchanges three messages [14] and causes significant computational overhead.

Our contribution. We provide protocols that would aid a monitoring setup as show in Fig. 1. Implementing our protocols on the system's side, and monitor's side enables monitoring where only one message is sent per observation round, the specification is kept secret, and the observable outputs of the system are kept secret. Instead of using secret sharing to construct PFE protocols for reactive functionalities, we propose novel protocols designed specifically for monitoring safety properties. Our protocols send a single message per round from the system to the monitor, reducing

computational cost while still ensuring privacy. The specification is given as a state machine with a next-state function that, in each round, updates the specification state based on an observation of the system. Additionally, the specification maintains a boolean output called *flag*. Our protocols ensure that the monitor learns about the monitored system only a single bit per round—the flag—which indicates whether or not the specification is satisfied for the prefix of the trace observed so far. In this way, the monitor can be convinced of the trace's correctness without knowing the input, output, nor internal state of the monitored system in any round, nor even the internal state of the specification itself.

We consider two security settings for which we provide privacy-preserving protocols for safety monitoring:

(a) [Open specification] The specification is not a secret and known to both parties.
(b) [Hidden specification] The specification is a secret and known only to the monitor.

Note that in setting (a), the monitored system knows which of its parts (which system inputs, which system outputs, and which internal system states) are being looked at by the specification, while in setting (b) it does not. Hence setting (b) is particularly interesting for large systems being monitored against small specifications.

Our privacy-preserving protocols are obtained by first converting the specification into a boolean circuit that computes a next-state function along with the flag function. For setting (a), where the specification is not a secret, we produce a protocol that is a modification of the classic MPC protocol known as Yao's garbled circuits [18, 43]. The intuition behind garbled circuits is that the monitored system encrypts each gate of the specification circuit and sends this encrypted value to the monitor. We modify the classical protocol to enable repeated computation while maintaining the secrecy of the specification state. For setting (b), where only the monitor knows the specification, inspired by recent advances in PFE, we provide novel protocols that compute reactive functionalities with low computational overhead. We build upon the seminal work of Katz and Malka [25] and the recent work of Liu, Wang, and Yu [31], which has built on other PFE-related works [7, 37]. Our protocols are designed so that the system sends only one message per round to the monitor. Although there is an initial setup phase that may involve multiple message exchanges, this is a one-time cost and does not affect the ongoing performance of the protocol during runtime. This is also helpful in monitoring situations where the hardware for bidirectional communication is unavailable.

We implemented our protocols to analyse the influence of several parameters involved in building such protocols. We use specifications described as register automata [19], which we convert to boolean circuits. Our experiments show the feasibility of our protocols when the circuit sizes are on the order of 10^5 for acceptable security parameters. Additionally, in our second protocol designed for hidden specifications, the time per round is influenced more by the size of the specification than by the size of the monitored system. This allows scalability to large system sizes as long as the specification remains small. Indeed, our protocol is significantly more scalable than those in recent related works [3, 41] which, although similar in motivation, address fundamentally different problems in distinct settings.

Related works. Recent work on privacy-preserving verification has largely focused on static settings. A wide range of verification paradigms have also been shown to be amenable to cryptographic techniques: for example, verifying resolution proofs in zero knowledge [32], solving SAT formulas [33], matching strings against regular expressions [34], checking string and regular expression equivalence [29], and model-checking CTL specifications [24]. In contrast, our work targets runtime verification, where the system's data evolves dynamically and must be processed incrementally at each step.

With a similar motivation of monitoring specifications in a privacy preserving manner, Banno et al. [3], and later Waga et al. [41] provided algorithms for oblivious online monitoring for Linear Temporal Logic (LTL) specifications [39] and Signal Temporal Logic (STL) [35] specifications, respectively using Fully Homomorphic Encryption (FHE). Although at first glance Banno et al. and Waga et al. may appear to solve the same problem, the settings for ours and their problems are different. Their main objective is to ensure monitoring of specifications where the computation can be outsourced to a second party, or an external server. Further, in their setting, this server knows the specification, however learns neither the system's observable outputs, nor if the specification is satisfied. Only the system itself learns if the specification is satisfied.

Due to our focus on enabling privacy-preserving third-party monitoring, in our setting, it is imperative that the party that knows the specification (monitor) can also learn if this specification is being satisfied, but not the system's observable outputs. Their protocol would therefore not be an appropriate solution in our setting, since it cannot be modified to make the second party that knows the specification learn whether the specification is satisfied, without also learning the system's observable outputs. On the other hand, both the related works deal with malicious Systems—a setting our protocol does not extend to, and hence our protocols would not be a appropriate solution in their setting either.

In terms of specifications, their work handles specifications represented using temporal logic that is later converted into finite state automata. We highlight that we consider circuits to described our specifications, and this representation is general enough that it handles all sequential specifications, including LTL, STL, or finite state automata, much more succinctly.

Since our work and their work deal with different settings, it is not ideal to compare either work with each other directly. However, since certain specifications considered by them can be of interest to us and vice-versa, we run our protocols on their specifications and extrapolate the time taken by theirs on ours in Section 4, where we show that our protocol handles all specs in the work of Bano et al. [3] within a few hundred milliseconds.

Our protocols for privacy-preserving runtime verification draw inspiration from several works on two-party computation [18, 42], and more specifically private function evaluation [7, 25, 31, 37]. Our Hidden Specification Protocol draws specifically on the recent work of Liu, Wang, and Yu [31]. In their work they provide a way to repeatedly compute the same function several times under standard IND-CPA and DDH assumptions against *covert adversaries*, a more adversarial setting than ours. However, Liu et al's protocol does not allow for hiding an internal monitor state (reactive functionalities).

2 Privacy-preserving monitoring

We describe our setting under which we perform privacy-preserving monitoring. We assume that there are two parties: the System (S) and the Monitor (M). We assume the two parties are *semi-honest*, that is, they follow the protocol, but might examine the transcripts to learn more information, as opposed to malicious parties that deviate from the protocol to obtain information.

System's observables. We assume that the System holds data that is represented as a string that is s-bits long. The System's observable sequence is temporal, and therefore, at each round r, the data output by the System is represented by $\sigma[r]$, and we use σ to represent the ℓ-length sequence $\sigma[1], \ldots, \sigma[\ell]$ of data.

Monitor's input. The monitor has a monitorable specification $\Phi \subseteq (\{0,1\}^s)^\omega$. Although other monitoring specifications can be considered [4], we consider the most important ones: safety specifications [27]. We assume that the specification Φ is represented as a state machine, that is, it starts at a state represented by $\mu[1]$, and at every round r where the observable output of the System is $\sigma[r]$, the Monitor's state is updated by a *deterministic* function $\mu[r + 1] = \text{nextstate}(\sigma[r], \mu[r])$. The Monitor has a function $\text{flag}(\sigma[r], \mu[r])$ that is 1 if a prefix is not in the specification, and 0 otherwise. We assume that such specifications are described as a circuit C that encodes both functions nextstate and flag. We call an *initialised circuit* as the pair of circuit and initial monitor state $(C, \mu[1])$.

Ideal settings. We describe the ideal settings with a trusted third party that our protocols must emulate. We describe two settings: one where the Monitor's specification (and therefore the circuit C representing it) is not private, and another where C is also kept private. If the specification is known to both parties, the Monitor hands over $\mu[1]$ to the trusted third party in the first round. If the specification is secret from the System, the Monitor also hands over $C = (\text{nextstate}, \text{flag})$. At every round r, the System hands over observable system output $\sigma[r]$ to this trusted third party. The trusted third party computes and returns $\tau[r] = \text{flag}(\mu[r], \sigma[r])$ and internally stores the value $\mu[r + 1] = \text{nextstate}(\mu[r], \sigma[r])$. These two settings are described pictorially in Figs. 2 and 3.

2.0.1 A monitoring protocol.
A monitoring protocol is a pair of instructions $\pi = (\pi_{\text{Sys}}, \pi_{\text{Mon}})$, one for the System and one for the Monitor, such that at each round r, the Monitor and the System send messages to each other as dictated by the protocol. A protocol has a round 0, which we refer to as the *setup phase*. Furthermore, in a fixed round, the only messages sent according to the protocol are from the System to the Monitor. The Monitor (optionally) responds with "proceed" or "terminate". We add the additional restriction of only one message per round of the monitoring protocol, since the process of monitoring needs to be relatively lightweight and not involve several exchanges within each round.

We first define the correctness of a monitoring protocol $\pi = (\pi_{\text{Sys}}, \pi_{\text{Mon}})$.

Definition 2.1 (Correctness of monitoring protocols with semi-honest parties). A protocol $\pi = (\pi_{\text{Sys}}, \pi_{\text{Mon}})$ is a *correct monitoring protocol* if for any specification described by an initialised circuit
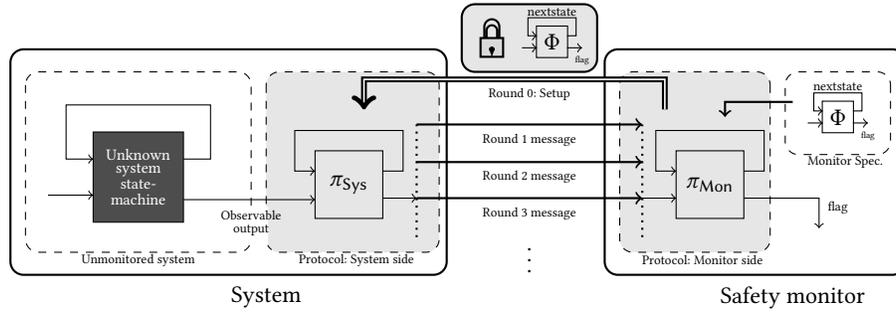
Figure 1: The architecture of a system and a safety monitor with a privacy-preserving protocol: $(\pi_{\text{Sys}}, \pi_{\text{Mon}})$

$(C, \mu[1])$, System sequence σ, and security parameter n, the output of the Monitor computed in an execution of the protocol π is equal to the output of the Monitor in the ideal setting on the same inputs with high probability, that is, probability $> \left(1 - \frac{1}{n^d}\right)$ for any fixed $d \in \mathbb{N}$.

Secure monitoring protocol. We define privacy of a monitoring protocol based on a comparison between the real and the ideal setting defined below. This definition of privacy compares the transcript of each party during the execution of a protocol with simulated transcripts generated by a simulator—a probabilistic polynomial-time machine—that has access only to one party's inputs and outputs. The simulator operates in an ideal world that involves a trusted third party performing the computations and delivering the results securely, thus ensuring privacy by definition. A protocol is considered secure if the real transcript and the simulated transcript are "indistinguishable." Since the protocols are probabilistic, this indistinguishability applies to the distributions of the transcripts.

In our ideal-world setting for the hidden spec protocol, the Monitor hands over the circuit of the specification to the trusted third party. This is kept secret from the System. The trusted third party carries out the task of evaluating if the System's input satisfies the specification and, by definition, does not leak the specification to the System. Indeed, in the real-world setting, trusted third parties don't exist, and our protocol ensures that despite the lack of trusted third parties, the specification circuit remains secret, as is proved in the appendix of our paper, by showing that the real world is "indistinguishable" from the ideal world in which trusted third parties exist.

Real view. The *view of the Monitor on protocol π*, written as $\text{view}^{\pi}_{\text{Mon}}(x_M, \sigma, 1^n)$ for an execution of a protocol π on inputs x_M of the Monitor and System observable output sequence σ is defined as a tuple consisting of the Monitor's input, the internal random bits that were used, and the messages $m_1, \ldots, m_\ell$ received by the Monitor, during the protocol execution, where m_j is the jth message.

Ideal view. The *simulated or ideal view of the monitor on protocol π*, written as $\mathcal{S}^{\text{IDEAL}}_{\text{Mon},\pi}(x_M, \sigma, 1^n)$, is a transcript generated by a simulator $\mathcal{S}_{\text{Mon}}$ (a probabilistic polynomial-time machine) that has access only to the inputs and outputs received by the Monitor in the ideal setting. We drop π from the subscript, if the protocol is clear from

context. The view $\text{view}^{\pi}_{\text{Sys}}(x_M, \sigma, 1^n)$ and the ideal view $\mathcal{S}^{\text{IDEAL}}_{\text{Sys}}$ of the System are defined in a similar way.

Definition 2.2 (Secure monitoring with semi-honest parties). A protocol $\pi = (\pi_{\text{Sys}}, \pi_{\text{Mon}})$ is a *secure monitoring protocol without specification hiding* for a specification represented by a circuit C if there are simulators (probabilistic polynomial time machines) $\mathcal{S}_{\text{Mon}}$ and $\mathcal{S}_{\text{Sys}}$ such that for any initial monitor states $\mu[1]$, System's observable sequences and security parameters $n \in \mathbb{N}$, we have

$$\left\{\mathcal{S}^{\text{IDEAL}}_{\text{Mon}}(\mu[1], \sigma, 1^n)\right\}_{\mu[1],\sigma} \overset{c}{\equiv} \left\{\text{view}^{\pi}_{\text{Mon}}(\mu[1], \sigma, 1^n)\right\}_{\mu[1],\sigma}$$

$$\left\{\mathcal{S}^{\text{IDEAL}}_{\text{Sys}}(\mu[1], \sigma, 1^n)\right\}_{\mu[1],\sigma} \overset{c}{\equiv} \left\{\text{view}^{\pi}_{\text{Sys}}(\mu[1], \sigma, 1^n)\right\}_{\mu[1],\sigma}.$$

A protocol $\pi = (\pi_{\text{Sys}}, \pi_{\text{Mon}})$ is a *secure monitoring protocol with specification hiding* if there are simulators (probabilistic polynomial time machines) $\mathcal{S}_{\text{Mon}}$ and $\mathcal{S}_{\text{Sys}}$ such that for all initialised circuits $(C, \mu[1])$, System observable sequence σ, and security parameters $n \in \mathbb{N}$, we have

$$\left\{\mathcal{S}^{\text{IDEAL}}_{\text{Mon}}((C, \mu[1]), \sigma, 1^n)\right\}_{(C,\mu[1]),\sigma} \overset{c}{\equiv}$$
$$\left\{\text{view}^{\pi}_{\text{Mon}}((C, \mu[1]), \sigma, 1^n)\right\}_{(C,\mu[1]),\sigma}$$

$$\left\{\mathcal{S}^{\text{IDEAL}}_{\text{Sys}}((C, \mu[1]), \sigma, 1^n)\right\}_{(C,\mu[1]),\sigma} \overset{c}{\equiv}$$
$$\left\{\text{view}^{\pi}_{\text{Sys}}((C, \mu[1]), \sigma, 1^n)\right\}_{(C,\mu[1]),\sigma}.$$

Not that we use the symbol $\overset{c}{\equiv}$ to denote the standard definition of computational indistinguishability [9]. We draw a pictorial representation of the ideal setting of a monitoring protocol. The simulator for each party resides in this is ideal setting, where each party has only the information it receives from the trusted third-party.

3 Protocols for privacy-preserving monitoring

Warm up–Yao's garbling with one gate. The most fundamental tool used in secure two-party computation is attributed to Yao and was dubbed Yao's garbling by Goldreich, Micali, and Wigderson [18]. We first describe a toy-version of the problem posed by Yao. Consider two parties A and B. Can we have a secure protocol where party A has two bits, and B wants to know the output of gate G computing a Boolean function over two bits, where gate G is known to both parties? Yao's garbling produces a simple solution to this as follows.
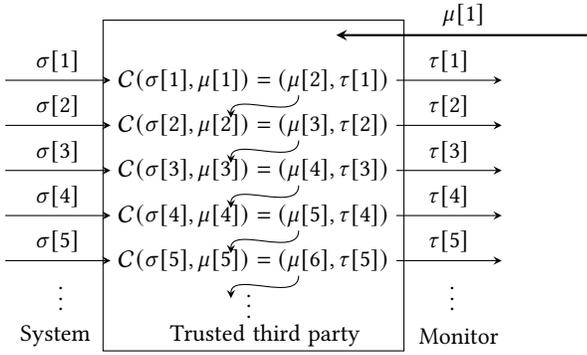
Figure 2: Ideal setting with a trusted third party for monitoring where the specification is *not* a secret and circuit C is known to all.
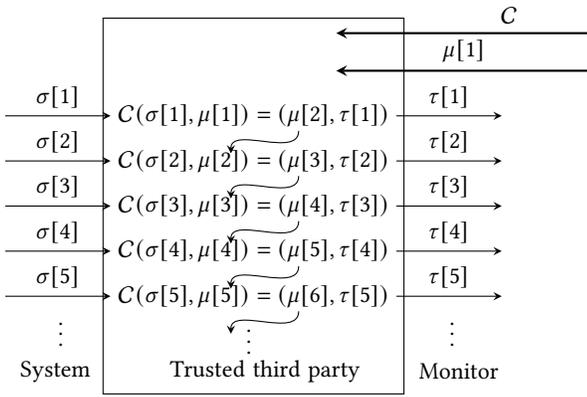

Figure 3: Ideal setting with a trusted third party for monitoring where the specification is a secret.

Party A starts by randomly generating strings of a fixed length L^0, L^1, R^0, R^1, S^0 and S^1. The strings L^0, L^1 correspond intuitively to each value 0 and 1 taken by the left input wire of the gate, respectively. Similarly R^0 and R^1 correspond to the values taken by the right input wire, and S^0 and S^1 to the output wire of the gate taking values 0 and 1, respectively. After the labelling step, Party A *encrypts* the label of the output of G using keys that correspond to the input. So, if party B had keys that corresponds to input $(0, 1)$, then it can only open the output that would represent $G(0, 1)$. More formally, party A calls a subroutine **encYao**$_G$ that generates the *garbled gate* which consists of four cipher-texts as follows

$$\textbf{encYao}_G\left([L^0, L^1], [R^0, R^1], [S^0, S^1]\right) \coloneqq$$
$$\left\{\text{Enc}_{L^\alpha, R^\beta}\left(S^{G(\alpha,\beta)}\right)\right\}_{\alpha,\beta \in \{0,1\}} \quad (\$)$$

and sends it to party B, but the encrypted messages are sent in random order. That is, if gate G was an *AND* gate, then the garbled gate would be a random order of the elements $\{\text{Enc}_{L^0, R^0}(S^0), \text{Enc}_{L^1, R^0}(S^0), \text{Enc}_{L^0, R^1}(S^0), \text{Enc}_{L^1, R^1}(S^1)\}$. Party A also sends $\langle S^0, 0\rangle$ and $\langle S^1, 1\rangle$ to indicate to B that S^0 corresponds to bit 0 and S^1 to bit 1. If party

A's input for the left and the right gate are $\ell, r \in \{0, 1\}$, then she also sends the random labels L^ℓ and R^r.

Party B then uses L^ℓ and R^r as keys to open the four ciphertexts; with very high probability, only one of the four will open, which corresponds to exactly $S^{G(\ell,r)}$. If $S^{G(\ell,r)} = S^0$, he concludes $G(\ell, r) = 0$, and 1 if $S^{G(\ell,r)} = S^1$.

Now, consider the same situation with a gate G over two bits, however, one bit of input is known to party A and the other bit is known to party B. Can we modify the above protocol to ensure that party B learns $G(\alpha, \beta)$ without learning α, where α is party A's input and β is party B's input?

Oblivious Transfer. The oblivious transfer functionality ensures that for two parties where one party, say party A, has two strings $x_0, x_1 \in \{0, 1\}^n$ and the other party, that is party B has one bit $\beta \in \{0, 1\}$, OT transfers the bit x_β to party B, without revealing to A the bit β, or the bit $x_{1-\beta}$ to party B. The functionality OT is defined as a functionality from $\{0, 1\}^{2n} \times \{0, 1\} \mapsto \{0, 1\}^n$, where $\text{OT}((x_0, x_1), \beta) = x_\beta$. Secure protocols for OT have been known since the 1980s, with the earliest forms proposed by Rabin [40], and later improvements and alternative protocols by Killian [26], and also by Bellare and Micali [6] and several others.

Using a protocol for the functionality Oblivious transfer (OT) as a sub-protocol, we can now modify the previously described protocol for garbling circuits when party B holds bit β. The garbling protocol proceeds as follows. Party A similarly finds labels L^0, L^1, R^0, R^1, S^0, and S^1 corresponding to 0 or 1 for the wires, and prepares the garbled gates as described in the previous step. After this, party A and party B run a protocol for Oblivious transfer where A has the labels for the input wire corresponding to B's input β, say R_0 and R_1, and B has the input bit. At the end of the OT protocol, B would receive R_β. Later, party A also sends L_α. This way, out of the four ciphertexts with the keys L_α and R_β, party B can only open the one ciphertext that contains the key $S^{G(\alpha,\beta)}$. He matches this string obtained with S^0 or S^1, both received from A.

Yao's garbling with a circuit. Consider the same problem, however, instead of just a simple gate G, it is a circuit C that party B wants to use to evaluate on party A's input. Then for each wire W in the circuit, party A similarly prepares random keys to represent the value W^0 and W^1. Whenever an output wire feeds into an input for a gate, party A uses the same random keys for the output and input wire. For each gate G in the circuit C, party A computes **encYao**$_G$ using the corresponding gate's inputs and outputs wire labels. Party A further sends the labellings generated for the output wire along with whether they correspond to 0 or 1 to B and the input labels of the wires which correspond to her input.

Party B can use the keys corresponding to the input of party A and unlock the gates to learn the keys to the next gates in a bottom-up manner and work through the circuit until he obtains the keys corresponding to the output wires. Then party B can match the keys with the corresponding values sent by A.

Conventions and notations for protocols. We describe our protocol for monitoring, which is a reactive functionality. Any description of a specification is converted into the circuit described below.

- *Circuit size.* Both the System and the Monitor agree that there are c many gates in the circuit C describing the specification.

The circuit has $s + m$ many inputs where the first m of the inputs correspond to the Monitor's state and the rest s to the System's input, which are the observable output. There are $m + 1$ output wires.

- *Circuit structure.* We assume that every gate is a NAND gate with exactly two wires that feed in and one that feeds out. We often use the terms *left* and *right* feed-in wire to differentiate between such wires for a fixed gate. We assume that any wire that is an output of a gate that is also an output wire of the circuit C does not connect back to any feed-in wires.

Wires. As discussed, there are two kinds of wires: *feed-out* and *feed-in* wires. The idea is that feed-in wires *feed into a gate* and feed-out wires *feed out of a gate*. Every feed-out wire can be connected to zero or more feed-in wires, but every feed-in wire is connected to exactly one feed-out wire; for this matter, we also take every input wire to circuit C to be a feed-out wire.

Naming the wires. There are $I = 2c$ feed-in wires (two feeding into each gate) and $c + s + m$ feed-out wires (one feed-out wire for each gate, and $s+m$ input wires of the circuit). We use $\iota_1, \iota_2, \ldots, \iota_I$, to represent the feed-in wires and use $\omega_1, \omega_2, \ldots, \omega_{c+s+m}$ to represent the feed-out wires. Among these feed-out wires, $m + 1$ wires are circuit output wires and the other $O = c + s - 1$ feed-out wires are not circuit-output wires. The first $m + s$ of the feed-out wires $\omega_1, \ldots, \omega_{m+s}$, represent the input wires to the circuit. The first m corresponds to the Monitor's input and the next s, the System's input to the protocol.

Gates. We call the c gates $G_1, G_2, \ldots, G_c$. The gate G_j has ι_{2j-1} as its left feed-in wire and ι_{2j} as its right feed-in wire. For each $j \in \{1, \ldots, c\}$, the wire ω_{m+s+j} denotes the feed-out wire of gate G_j. The feed-out wires $\omega_{m+s}, \ldots, \omega_{O+m+1}$ denote the output wires of the circuit. Therefore, the last $m + 1$ feed-out wires of gates $G_{c-m-1}, \ldots, G_c$ correspond exactly to the last $m + 1$ output wires $\omega_{O+1}, \ldots, \omega_{O+m+1}$, respectively. The output wires of the monitor state that are used for feed-back into the next round, are represented by $\omega_{O+1}, \omega_{O+2}, \ldots, \omega_{O+m}$ and the output wire corresponding to flag is represented by ω_{O+m+1}. See the black text in Fig. 4 for a pictorial representation of the names of the wires.

3.1 First protocol - Monitoring without specification hiding

We first provide a conceptually simpler protocol for when the specification and the circuit C representing it are known to both parties. The internal state computed by the circuit is still kept secret. The protocol is similar to Yao's garbling with a circuit described earlier. The main modification here is to reuse the labels for the output wires from one round for the Monitor's state for the input component of the Monitor's state in the next round. This operation is described in line 2 of the protocol. To obtain the output, the Monitor uses the keys to unlock the circuits from bottom-up, similar to Yao's protocol.

The following theorem shows that Open Specification Protocol is correct and secure, assuming that the encryption is secure under the chosen plaintext attack (CPA) [8].

Open Specification Protocol Secure monitoring without specification hiding

Require: Both parties the Monitor and the System have circuit C as described and agree on a security parameter n. Additionally, the Monitor holds an m bit-string $\mu[1] = \mu_1[1] \cdot \mu_2[1] \cdots \mu_m[1]$ that correspond to the valuation of the initial state, and at each round r, the System holds a s-bit string $\sigma[r] = \sigma_1[r] \cdot \sigma_2[r] \ldots \sigma_s[r]$ that corresponds to the current System state.

Ensure: At round r, the Monitor learns *only* the last output bit of the circuit C on the string $(\sigma[r], \mu[r])$, and neither party learns the value of $\mu[r]$ for $r > 1$.

1: S : For all out-going wires ω_i other than the wires corresponding to the input of the Monitor, i.e., $i \in \{m + 1, \ldots, c + s + m\}$, the System generates two random strings $w_i^0[r]$ and $w_i^1[r]$ corresponding to it.

2: S : For the out-going wire ω_i that also correspond to the m input wires (for all $i \in \{1, \ldots m\}$), the System acts depending on round r:
- **for** $r = 1$, the system chooses two string $w_i^0[1]$ and $w_i^1[1]$ uniformly at random, and
- **for** $r > 1$, it reuses the labels of the feedback wires from the previous round i.e. $w_i^0[r] \leftarrow w_{O+i}^0[r - 1]$ and $w_i^1[r] \leftarrow w_{O+i}^1[r - 1]$.

3: S: For each in-going wire ι_i (for all $i \in \{1, \ldots, 2c\}$), the System assigns the labels from the output wire ω_j that is connected to it, i.e., $u_i^0[r] = w_j^0[r]$ and $u_i^1[r] = w_j^1[r]$ if the output wire ω_j is connected to the feed-in wire ι_i.

4: S: The System computes for each gate G_j, $\mathbf{encGG}_j[r]$ and sends the list $\mathbf{encGG}_j[r]$, the labels of the feed-in wires to open the gates for the values corresponding to each bit in $\sigma[r]$, that is, $w_1^{b_1}[r], \ldots, w_s^{b_s}[r]$ where $b_i = \sigma_i[r]$, and both the labels of the output flag bit w_{O+m+1}^0 and w_{O+m+1}^1, and $\mathbf{encGG}_j[r]$ is

$$\mathbf{encGG}_j[r] = \mathbf{encYao}_{G_j} \begin{pmatrix} \left[u_{2j-1}^0[r], u_{2j-1}^1[r] \right], \\ \left[u_{2j}^0[r], u_{2j}^1[r] \right], \\ \left[w_{m+s+j}^0[r], w_{m+s+j}^1[r] \right] \end{pmatrix}$$

5: S, M : For $r = 1$, the System also uses oblivious transfer to send the labels $w_1^{c_1}[1], \ldots, w_m^{c_m}[1]$ where c_i is the ith bit of $\mu[1]$ representing the Monitor input.

6: S : the Monitor ungarbles the circuit using previously obtained Monitor-state output labels and System's new labels to compute the Monitor-state and flag labels.

THEOREM 3.1. *Assuming the chosen encryption **Enc** is secure under the CPA model and the oblivious transfer protocol is secure in the presence of semi-honest adversaries, Open Specification Protocol is a* correct and secure *monitoring protocol without specification hiding, when both parties are semi-honest, and the number of rounds is a fixed polynomial in the security parameter.*
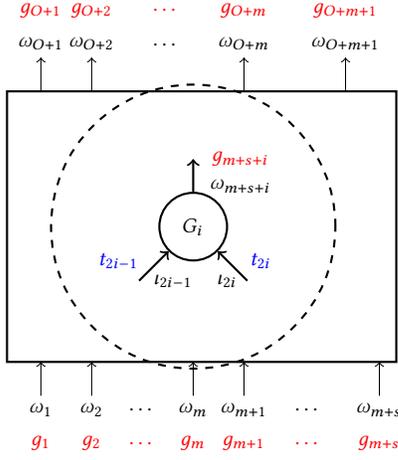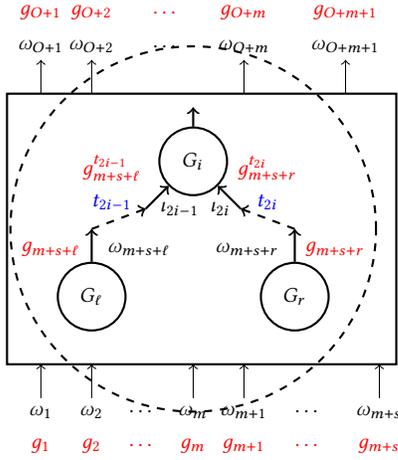
Figure 4: Base labels: feed-out wires.

Figure 5: Base labels: feed-in wires

3.2 Second protocol - Monitoring with specification hiding

The challenge with designing a protocol where the circuit must also be hidden is that the System cannot come up with the labels for the gates in the circuit, since this requires that the System know the topology of the circuit. As a first step, since the topology of the circuit must be kept secret, we assume that the circuit contains only NAND gates. We provide a protocol, which is a modification of the protocol of Liu et al. [31]. In our protocol, the Monitor helps the System to come up with the labellings for each gate and also obfuscates the topology of the circuit in this process. The Monitor does so by using a cyclic group $\mathbb{G}$ of prime order q where the Decisional Diffie-Hellman (DDH) assumption holds. In a cyclic group, for any non-unitary element g, and for two values in $a, b \in \mathbb{Z}_q$, it holds that $(g^a)^b = (g^b)^a$.

The Monitor assigns *base labels* to the feed-out wire of each gate G using a randomly chosen element, say g_G, from the group $\mathbb{G}$. If the feed-out wire of the gate G connects to some feed-in wire, then

the monitor randomly chooses exponent $t \in \mathbb{Z}_q$ for that feed-in wire and labels this wire using $(g_G)^t$.

More specifically, the Monitor selects the base labels using randomly generated group element g_i (represented in red in Fig. 4) for each feed-out wire ω_i. Then, for each feed-in wire ι_j, the Monitor also chooses an exponent t_j (represented in blue in Fig. 4). Finally, the Monitor computes and sends the base labels for each feed-in wire, as follows. For a gate G_i with feed-in wires ι_{2i-1} and ι_{2i} that are connected to feed-out wires from gates G_ℓ and G_r (as show in Fig. 5), the monitor uses as labels $g_{m+s+\ell}^{t_{2i-1}}$ and $g_{m+s+r}^{t_{2i}}$ as the base labels. Note that $g_{m+s+\ell}$ and g_{m+s+r} are the base-labels of the corresponding feed-out wires of gates G_ℓ and G_r.

The monitor sends three base labels for each gate: two for the feed-in wires and one for the feed-out. Roughly, the DDH assures that "random exponents of group elements" cannot be distinguished from "random group elements". Since DDH assumption holds, we can also show that given n group elements $g_1, g_2, \ldots, g_n$ as well as some labels $g_{x_1}^{t_1}, g_{x_2}^{t_2}, \ldots, g_{x_n}^{t_n}$, the System cannot tell which of the $g_{x_i}^{t_i}$s is obtained by exponentiating which of the g_i, thus successfully obfuscating the circuit topology.

Finally, the System uses these base labels to prepare the labels of the wires that correspond to 0 and 1. This is done by choosing one exponent to correspond to the value 0 and one exponent to correspond to 1, say α_0 and α_1. For the base label g, the System would then label each $(g)^{\alpha_0}$ and $(g)^{\alpha_1}$ corresponding to bits 0 and 1, respectively. Feed-in wire labels are of the form $(g^t)^{\alpha_i}$, which is equal to $(g^{\alpha_i})^t$. So, by knowing the label g^{α_i} (obtained by opening a garbled gate, or from the message of the System), the Monitor can compute $(g^{\alpha_i})^t = (g^t)^{\alpha_i}$.

Therefore, the Monitor obtains the key to open future gates by simply exponentiating the label obtained from the ungarbling process of a preceding gate and exponentiating with an appropriate exponent t. The System, under the Decisional Diffie-Hellman (DDH) assumption stated below, cannot learn the topology only given such exponentiated group elements.

We show that our protocol is correct and secure under the DDH assumption over groups of prime order. Both message sizes and the time taken of our protocol is linear, that is, $O(c + s + m)$, assuming the security parameter is a constant.

ASSUMPTION 1 (DECISIONAL DIFFIE–HELLMAN ASSUMPTION [10, 15]). *In a cyclic group* $\mathbb{G} = \langle g \rangle$ *of prime order* $q \in \Theta(2^k)$ *for* $n \in \text{poly}(k)$, *where g is a generator for* $\mathbb{G}$, *the following probability distributions are computationally indistinguishable:* (g^a, g^b, g^{ab}), *where a and b are uniformly and independently at random chosen from* $\mathbb{Z}_q$, *and* (g^a, g^b, g^c), *where a, b and c are uniformly and independently at random chosen from* $\mathbb{Z}_q$.

THEOREM 3.2. *Assuming that the chosen encryption* **Enc** *is secure under the CPA model, the DDH assumption on group* $\mathbb{G}$ *holds, and the oblivious transfer protocol is secure in the presence of semi-honest adversaries, Hidden Specification Protocol is a* correct and secure monitoring protocol with specification hiding *when both parties are semi-honest, and the number of rounds is a fixed polynomial in the security parameter.*

The proof of correctness of our theorem is a simulation based proof that constructs intermediate indistinguishable transcripts by

Hidden Specification Protocol Secure monitoring with specification hiding

Require: Similar to Open Specification Protocol, however, only the Monitor knows the specification circuit C, but both parties agree on the number of gates c of the circuit, a security parameter n, and a group $\mathbb{G}$ of order $q \in \Theta(2^n)$.

Ensure: At round r, the Monitor learns *only* the last output bit of the circuit C at round r, and neither party learns the value of $\mu[r]$ for $r > 1$, and the System does not learn C.

1: **Setup.** Base labels of feed-out wires: (See Fig. 4)

    a. The Monitor picks random group elements $g_i \in \mathbb{G}$ for each feed-out wire ω_i for $i \in \{1, \ldots, O\}$ and sends the list $[g_1, g_2, \ldots, g_O]$.

    b. Further, the output wires, which correspond to the m feed-out wires $\omega_{O+1}, \ldots, \omega_{O+m}$ are also given the (same) group elements $g_1, g_2, \ldots, g_m$, respectively. The final output wire computing flag is represented by ω_{O+m+1} and is not assigned a group element during setup phase.

    Base labels of feed-in wires: (See Figs. 4 and 5)

    c. The Monitor further picks different exponents $t_i \xleftarrow{\$} \mathbb{Z}_q$ for each feed-in wire $i \in \{1, \ldots, I\}$, and finds the map $\pi : \{1, \ldots, I\} \mapsto \{1 \ldots, O\}$ (chosen uniformly at random) such that $\pi(i) = j$ iff the feed-out wire ω_j connects to the feed-in wire ι_i.

    d. The Monitor then computes $\ell_i = g_{\pi(i)}^{t_i}$ for every $i \in \{1, \ldots, I\}$ and creates list $L = [\ell_1, \ell_2, \ldots, \ell_I]$ and sends this list to the System. This assigns group element $g_{\pi(i)}^{t_i}$ to the feed-in wire.

2: **Labelling wires at round** $r \geqslant 1$

    e. For the first round, $r = 1$, the System picks random, distinct values $\alpha^0[r], \alpha^1[r] \xleftarrow{\$} \mathbb{Z}_q$. For subsequent rounds, $\alpha^0[r] \leftarrow \beta^0[r-1]$ and $\alpha^1[r] \leftarrow \beta^1[r-1]$.

    f. The System assigns values $w_j^0[r]$ and $w_j^1[r]$, which corresponds to the feed-out wire ω_j having value 0 and 1, respectively, to the feed-out wires for all $j \in \{1, \ldots, O\}$, $w_j^0[r] \leftarrow g_j^{\alpha^0[r]}$ and $w_j^1[r] \leftarrow g_j^{\alpha^1[r]}$.

    For all rounds r, the System also selects random values $\beta^0[r], \beta^1[r] \xleftarrow{\$} \mathbb{Z}_q$ and it computes the label of the feed-out wires for $j \in \{O+1, \ldots, O+m\}$ as $w_j^0[r] = g_{j-O}^{\beta^0[r]}$ and $w_j^1[r] = g_{j-O}^{\beta^1[r]}$.

    For the feed-out wire ω_{O+m+1} representing the output of flag, it assigns $w_{O+m+1}^0[r] \xleftarrow{\$} \mathbb{G}$ and $w_{O+m+1}^1[r] \xleftarrow{\$} \mathbb{G}$. The System labels the feed-in wires, for each $i \in \{1, \ldots, I\}$, $u_i^0[r] = \ell_i^{\alpha^0[r]}$ and $u_i^1[r] = \ell_i^{\alpha^1[r]}$.

    g. Proceed as in Steps 4., 5., and 6., in Open Specification Protocol, where the System garbles the gates and sends the desired keys, and Monitor ungarbles.

substituting elements of the transcript with random group elements. A key result required to prove indistinguishability is a corollary of a lemma from the work of Naor and Reingold [38] that is an equivalent representation of the DDH assumption in our proofs,

we prove the theorem. A similar lemma is also used in the work of Liu, Wang, and Yu [31, Lemma 3].

Lemma 3.3 ([38, Lemma 4.4]). *Assuming that the DDH assumption holds in a cyclic group* $\mathbb{G} = \langle g \rangle$ *of prime order* $q \in \Theta(2^k)$ *for* $n \in \text{poly}(\kappa)$, *given* n *randomly chosen elements from the group* $g_1, g_2, \ldots, g_n \xleftarrow{\$} \mathbb{G}$ *and* $n+1$ *randomly chosen exponents* $a, a_1, a_2, \ldots, a_n \xleftarrow{\$} \mathbb{Z}_q$, *we have that* $(g_1^a, g_2^a, \ldots, g_n^a)$ *is computationally indistinguishable from an n-tuple* $(g_1^{a_1}, g_2^{a_2}, \ldots, g_n^{a_n})$.

4 Experimental Evaluation

To test the feasibility of our protocols for monitoring, we developed an experimental C++ prototype[*] and performed experiments to evaluate the following key questions.

  (1) How do the measured requirements of both protocols change under varying security parameters ($n \geqslant 1024$ would be industrial standard),
    (a) in terms of time taken per round? (see Figs. 8 and 9)
    (b) in terms of message sizes per round? (see Figs. 6 and 7)
  (2) When the specification size (c) is fixed, but the size of System observables data (s) is large, how much do these measurements change for Hidden Specification Protocol? (see Figs. 11 and 12)

To answer these questions, we consider the following experiment scenarios:

  (1) An *access control system (ACS)* for an office building, where two types of employees, namely types A and B, enter or exit the building through a set of external doors. The ACS tracks the numbers of entries and exits for each type of employee and for each door. The Monitor keeps two variables cnt_A and cnt_B, where cnt_e denotes count of type-e employees currently in the building. At round r, for door i of the building, the Monitor receives input from the ACS, structured as follows: $\text{entered}_A^i[r]$, $\text{exited}_A^i[r]$, $\text{entered}_B^i[r]$, $\text{exited}_B^i[r]$. There are N doors, hence N such quadruples. Each entered_e^i denotes the number of type-e employees who have entered through door i of the building *since* the last round; exited values have a similar definition.

    Specification. Our specification for this system requires that the number of type A employees currently in the building is never less than type B employees; concretely, the flag function of the register machine activates if and only if $\text{cnt}_A < \text{cnt}_B$. Value of cnt_e updates with the following rule: $\text{cnt}_e \leftarrow \text{cnt}_e + \sum_i \left(\text{entered}_e^i[r]\right) - \sum_i \left(\text{exited}_e^i[r]\right)$. Each number is an unsigned integer of fixed bit-width W. The monitor only keeps track of employee count per type and needs $2W$ bits for Monitor state, whereas the input of the ACS to each round takes $4NW$ bits.

    To answer Question 2, we create another case where the ACS keeps track of the internal doors in the building as well (e.g., individual offices). We use N' to denote the number of internal doors. In this case, each ACS update takes $4(N + N')W$ bits, but the specification is still expressed over only $4NW$ bits.

---

[*]This prototype is accessible online at https://github.com/mahyari/ppm.
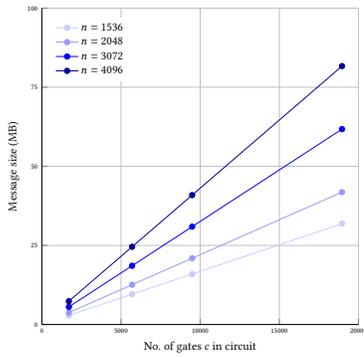
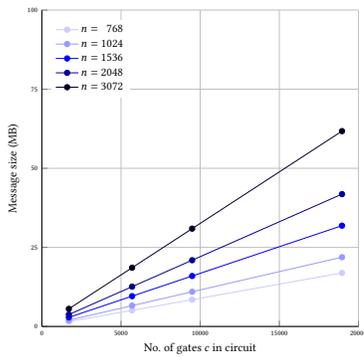**Figure 6: Message size vs gate count: Open Specification Protocol**

Figure 7: Message size vs gate count: Hidden Specification Protocol

(2) *The locks of a parallel program*, where every lock has at most one 'user' at any given time. Each lock provides a lock() and unlock() interface, and all the locks are initially 'unlocked'. The Monitor keeps track of all lock states $\text{lock}_1, \ldots, \text{lock}_N$, where N denotes the total number of locks in the system. Each lock_i can have value LOCK or UNLOCK. The Monitor, at round t, receives input from the lock system, structured as follows: $\text{request}_1[t], \ldots, \text{request}_N[t]$, where N denotes the number of locks, and each request can have value LOCK, UNLOCK, or SKIP.

Specification. The specification for this scenario requires that lock() or unlock() is never called twice in a row for any of the locks in the system. Concretely, at round r, we have: flag $\iff \bigvee_i (\text{request}_i[r] = \text{lock}_i)$. To update lock_i, we simply replace its value with $\text{request}_i[r]$. As the Monitor keeps track of all locks, it needs N bits for its state. Each request takes 2 bits to represent, and hence each update to the Monitor needs $2N$ bits.

Experiment results summary. To answer Question 1a, we plot the breakdown of execution times for Open Specification Protocol in Fig. 8 (scenario ACS), and Figs. 9 and 10 (scenarios ACS and locks) for Hidden Specification Protocol. For these cases, ACS only updates on external doors (i.e., $N' = 0$), and we select values for N among $\{10, 30\}$ and W among $\{16, 32\}$ to give us 4 different instances. For
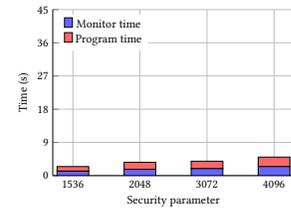
(a) $s = 640, c = 7127$
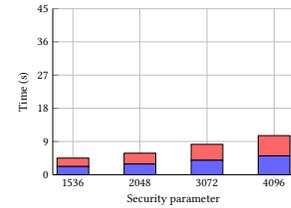
(b) $s = 1280, c = 14542$
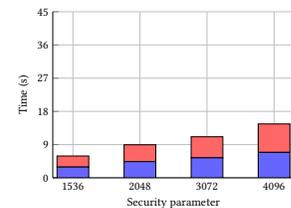
(c) $s = 1920, c = 20618$
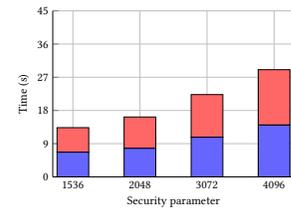
(d) $s = 3840, c = 42145$

Figure 8: Timings for the ACS scenario: Open Specification Protocol.

the Locks scenario, we consider the values $\{100, 300, 500, 1000\}$ for the parameter N. Open Specification Protocol relies only on random string generation instead of group operations and has symmetric garbling and ungarbling phases and the round times are lower and more uniformly distributed between System and Monitor. In the Hidden Specification Protocol, time is mostly spent on the System side, since System performs more group operations than Monitor in the garbling process; as the security parameter increases, this difference becomes more visible. The superlinear growth of round times also conforms with the complexity of group operations.

For Question 1b, we plot the breakdown of execution times for Open Specification Protocol in Fig. 6 and for Hidden Specification Protocol in Fig. 7. For the Locks scenario, we consider the values $\{100, 300, 500, 1000\}$ again for the parameter N that represents the number of locks. As expected, we only observe a linear correlation between message sizes and gate counts in both protocols.
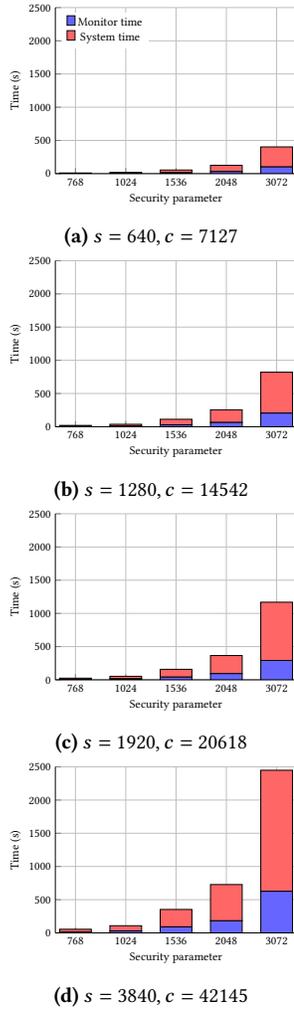
(a) $s = 640, c = 7127$

(b) $s = 1280, c = 14542$

(c) $s = 1920, c = 20618$

(d) $s = 3840, c = 42145$

Figure 9: Timings for the ACS scenario: Hidden Specification Protocol.

(a) $s = 200, c = 1705$

(b) $s = 600, c = 5700$

(c) $s = 1000, c = 9493$

(d) $s = 2000, c = 18959$
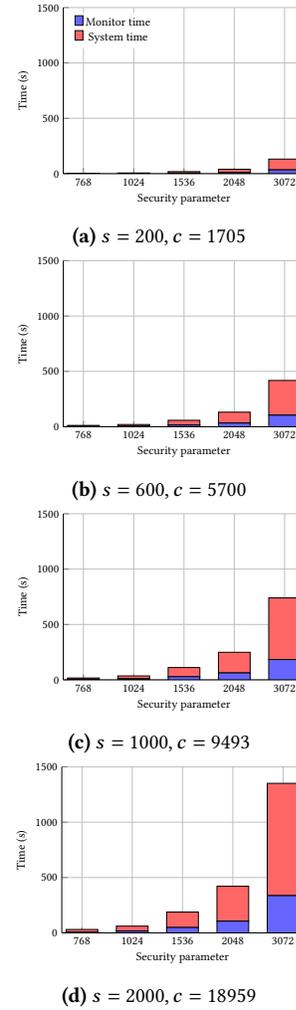
Figure 10: Timings for the Locks scenario: Hidden Specification Protocol.

Benchmark	Banno et al.	
	DFA Size	Time
MOD ($m = 500$)	500	~0.002 s
BGM (ψ_2), REVERSE	2885376	~24 s
BGM (ψ_2), BLOCK	11126	0.182 s
BGM (ψ_4), REVERSE	N/A	time-out
BGM (ψ_4), BLOCK	7026	0.049 s
ACS (Fig. 9a)	$\geqslant 2^{32}$	10 hours (estimated)
LOCKS (Fig. 10a)	$\geqslant 2^{300}$	10^6 years (estimated)

Table 1: Monitoring latency of a single event in the protocol of Banno et al. [3]
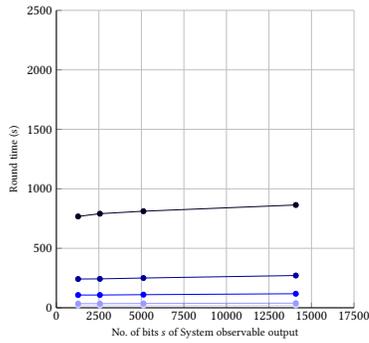
Benchmark	Hidden Specification Protocol	
	Circuit Size	Time for $n = 1024$
MOD ($m = 500$)	146	0.30 s
BGM (ψ_2)	118	0.24 s
BGM (ψ_4)	89	0.23 s
ACS (Fig. 9a)	7127	18.94 s
LOCKS (Fig. 10a)	5700	19.96 s

Table 2: Monitoring latency of a single event in Hidden Specification Protocol
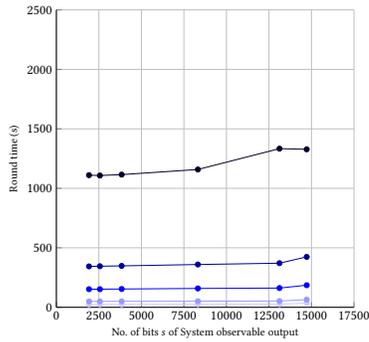
Finally, for Question 2, we plot Figs. 11 and 12, which shows how round time increases with increasing sizes of System's observable data, while keeping "relevant" System input size fixed and again consider parameters of N and W from $\{10, 30\}$ and $\{16, 32\}$,
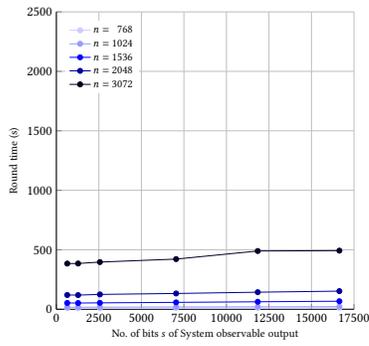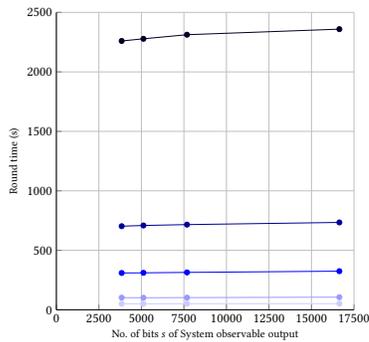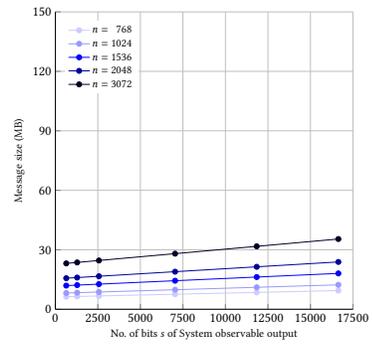
(a) $c = 7127$

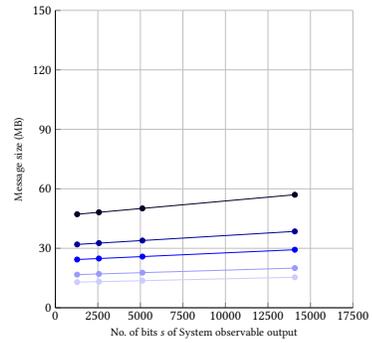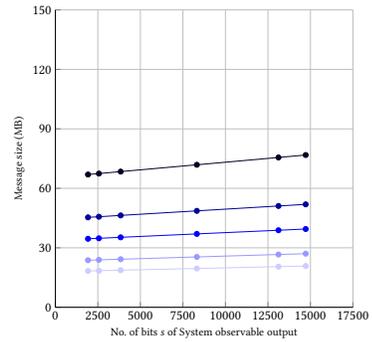(b) $c = 14542$

(c) $c = 20618$

(d) $c = 42145$

Figure 11: Timings for the ACS scenario, Hidden Specification Protocol; fixed specifications, increasing System observable output size.
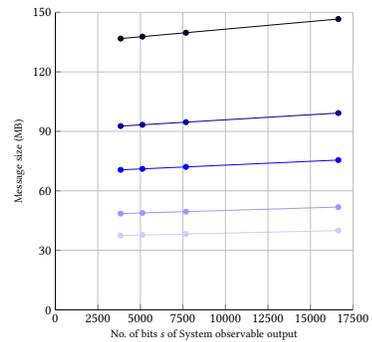

(a) $c = 7127$

(b) $c = 14542$

(c) $c = 20618$

(d) $c = 42145$

Figure 12: Message size ACS scenario: fixed specs, increasing System observable output size

respectively. However, to inflate System input size, we use increasing values for the parameter N' (no. of internal doors). Observe that the circuit size remains the same even while the specification considers increasing number of external doors. With the parameters we have used in our experiments, as System input increases, the time taken per round increases only marginally, as a result of the garbling phase's dominance in execution time (Fig. 11). The increase in message size is more observable, as one key is sent per each bit of the input (Fig. 12). However, even this growth is less steep than the growth of the message size when both circuit size and System observables are scaled proportionally as seen in Fig. 7.

We remark that all the monitoring latencies reported are for a single event rather than a trace.

As discussed in the related work section, the setting considered by Banno et al. [3] (and therefore also Waga et al [41]) is orthogonal to ours—each protocol is tailored to its specific context and not directly applicable to the other. However, some of their specifications they describe might be relevant to our setting, and vice versa. Therefore, we evaluated our protocol on several of their specifications and, conversely, estimated the performance of their protocol on the ACS and locks scenarios from our work.

Banno et al. consider two scenarios: DFA that counts number of 1s in its input modulus m (MOD) and Blood Glucose Monitoring (BGM). They also have two protocols REVERSE and BLOCK. We implemented the specifications for each scenario with the *highest* reported monitoring latencies in their work for either REVERSE or BLOCK, with the same security parameter as Banno et al. ($n = 1024$). For these values, our protocols take time that is in the order of magnitude of 100 milliseconds. We remark that their experiments were run on Intel Xeon Silver (32 cores and 64 threads), a superior hardware to ours. These times are summarised in Table 1.

To test our specifications against their protocol, we also estimated the time taken by their protocol on the ACS and Locks scenarios we designed. However, our scenarios cannot be directly specified as LTL expressions over the observable output alone to be used directly as an input into their protocol, since our description language is more expressive than LTL specification. Since their protocols converts LTL specifications into DFAs, we considered the size of the smallest possible DFAs accepting the ACS and Locks scenarios to estimate the running time. We then extrapolated the monitoring latency on ACS and the locks scenario from the fact that their protocol is linear in the size of the DFA, and scale from the other DFA instances provided in their work. These times are summarised in Table 2.

For both Tables 1 and 2, we use the same notation as their paper to refer to the LTL specifications (ψ_2 and ψ_4) in their work, obtained originally from related work on runtime verification for artificial pancreas [13].

The sizes of the formula considered in the experiments conducted by Waga et al. [41] are similar in terms of DFA sizes to those considered by Banno et al., and we therefore only restrict our comparison to Banno et al's work. It is reasonable to expect that the monitoring latency of both Waga et al's protocols on these specifications would also be in the same order of magnitude.

Execution pipeline. We write each specification as a synthesisable Verilog module, which describes the nextstate and flag functions

of our intended register machine. We then synthesise the equivalent circuit, which is what we will use throughout the protocol execution.

We spawn Monitor and System as separate processes that interact via message passing. Internally, we have modelled and implemented each party as a communicating transition system, with access to asynchronous communication channels. Each state of the transition system can perform read or write operations on a dedicated memory fragment. This design allows us to invoke a protocol within a protocol, as every state can internally keep track of the execution of another transition system, and proceed once the internal protocol is done. An example of this use case is our OT step in the initialisation phase of the Hidden Specification Protocol.

We use the GNU Multiple Precision Arithmetic (GMP) library for big-integer arithmetic, OpenSSL for hash functions, and ZeroMQ for asynchronous message passing. We also use the Yosys open synthesis suite for synthesising Boolean circuits from specifications. We pass a Verilog specification module to Yosys, with ABC [12] as synthesis back-end, to obtain a Boolean circuit equivalent, represented in the Berkeley logic interchange format (BLIF). Note that all optimisations on circuit size are also done by Yosys and ABC.

Cryptographic primitives. The DDH assumption holds in the *quadratic residue* group QR_q if q is a *safe prime* [10]. We use groups QR_q in our protocols, with values of q defined in RFC 2409, 3526 [16, 28]. The security parameter specifies the binary representation size of q. Similar to Huang et al. [21], we construct the symmetric encryption scheme for **EncYao** (see Eq. ($\$$)) using a hash function; precisely, for keys L and R and secret S, we have: $\text{Enc}_{L,R}(S) = \text{SHAKE-256}\,(L \parallel R) \oplus (S \parallel 1^{100})$, where $\cdot \parallel \cdot$ denotes string concatenation, and SHAKE-256 is an *extendable-output* hash function from the SHA-3 family; since group elements (hence, their representation) can be arbitrarily large, we needed a hash function with arbitrarily large output. Note that, contrary to the parametrised security level for the groups we use, SHAKE-256 has a fixed security level of 256 bits; however, this does not impose any practical vulnerabilities and, therefore, is practically secure. The 100-bit constant padding at the end of the secret is necessary for the decryption phase (gate un-garbling), to detect the correctly decrypted value. We use the simple OT protocol introduced by Bellare and Micali [6] in both protocols.

All experiments were run on a personal computer with an Intel Core i5-1235U processor, 16 GB of memory, running Linux Mint 21.3. Both System and Monitor processes were spawned in parallel, and bound to `localhost` for communication. We use a timeout of one hour per protocol round, excluding initialisations that take place only in the first round. As a source of true randomness, we periodically read from the file `/dev/urandom`; we buffer a fixed number of such values in program memory, in order to perform fewer file I/O operations.

Remark. Unlike Hidden Specification Protocol, which requires circuits with a single type of binary gate, Open Specification Protocol supports multiple gate types, enabling the use of optimised garbling techniques [1, 44]. Further, even Hidden Specification Protocolcan be made more efficient if the number of gates of each type is known to all, while still ensuring the circuit topology is not

known. This opens potential avenues for optimisation. Additionally, our experiments showed a 50% average reduction in circuit size when ABC utilised all basic gates instead of only NAND gates, which could contribute to further speed-up.

5 Outlook

We took a first step toward privacy-preserving monitoring by proposing protocols that are correct and secure. Our experiments demonstrate an increase in both the message length and the protocol's overhead with respect to the security parameter, demonstrating a trade-off between privacy and efficiency.

While the levels of privacy provided by our protocols are sufficient for many real-world applications, they fall short of the requirements in highly privacy-sensitive settings. First of all, we only focus on cryptographic privacy in our work, and further, we only assume semi-honest parties in our current work. Extending these protocols to protect against actively malicious parties—those who intentionally deviate from the protocol—may be computationally expensive. A potential compromise is to consider *covert systems*, as proposed by Aumann and Lindell [2], where adversaries that deviate from the protocol are caught with a positive probability. For monitoring applications, repeated interactions increase the likelihood of catching cheating agents across multiple rounds, ultimately approaching probabilities close to 1.

Even for the setting of semi-honest parties, we believe that our protocol could be enhanced by optimising the number of gates that represent the specification (see Fig. 11). Heuristics can be employed to reduce circuit size, but an alternative approach is to relax the specifications—either in terms of soundness or completeness—depending on the specific requirements, to enable encoding with smaller circuits. Another direction to improve the performance of our protocols is to parallelise the protocols. The process of garbling gates is inherently parallelisable for both parties, particularly for the system. Similarly, the monitor's task of ungarbling can also be parallelised, with the primary bottleneck being the depth of the circuit. Consequently, finding circuits with lower depth, even if it means increasing the number of gates, could enable faster parallelised algorithms.

Our protocol works for specs described by register automata, or using Yosys. It would be future work to integrate it with state-of-the-art monitoring tools such as BeepBeep [11], DejaVu [20], or MonPoly [5]. Lastly, our current protocols assume that the monitor and the monitored system are single entities, and that the monitor relies on a linear order of observations. Developing privacy-preserving monitoring protocols for scenarios where the system and monitor are distributed is an interesting research challenge.

6 Acknowledgements

References

[1] Benny Applebaum. 2016. Garbling XOR Gates "For Free" in the Standard Model. *Journal of Cryptology* 29, 3 (01 Jul 2016), 552–576. doi:10.1007/s00145-015-9201-9

[2] Yonatan Aumann and Yehuda Lindell. 2010. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. *J. Cryptol.* 23, 2 (2010), 281–343. doi:10.1007/S00145-009-9040-7

[3] Ryotaro Banno, Kotaro Matsuoka, Naoki Matsumoto, Song Bian, Masaki Waga, and Kohei Suenaga. 2022. Oblivious Online Monitoring for Safety LTL Specification via Fully Homomorphic Encryption. In *Computer Aided Verification - CAV*. Springer International Publishing, 447–468.

[4] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. 2018. *Introduction to Runtime Verification*. Springer International Publishing, Cham, 1–33. doi:10.1007/978-3-319-75632-5_1

[5] David A. Basin, Felix Klaedtke, and Eugen Zalinescu. 2017. The MonPoly Monitoring Tool. In *RV-CuBES*. https://api.semanticscholar.org/CorpusID:33336626

[6] Mihir Bellare and Silvio Micali. 1989. Non-Interactive Oblivious Transfer and Applications. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference (Lecture Notes in Computer Science, Vol. 435)*, Gilles Brassard (Ed.). Springer, 547–557. doi:10.1007/0-387-34805-0_48

[7] Muhammed Ali Bingöl, Osman Biçer, Mehmet Sabir Kiraz, and Albert Levi. 2019. An efficient 2-party private function evaluation protocol based on half gates. *Comput. J.* 62, 4 (2019), 598–613.

[8] Alex Biryukov. 2011. *Chosen Plaintext Attack*. Springer US, Boston, MA, 205–206. doi:10.1007/978-1-4419-5906-5_557

[9] Manuel Blum and Silvio Micali. 1982. How to Generate Cryptographically Strong Sequences of Pseudo Random Bits. In *23rd Annual Symposium on Foundations of Computer Science FOCS*. IEEE Computer Society, 112–117. doi:10.1109/SFCS.1982.72

[10] Dan Boneh. 1998. The Decision Diffie-Hellman problem. In *Algorithmic Number Theory*, Joe P. Buhler (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 48–63.

[11] Mohamed Recem Boussaha, Raphaël Khoury, and Sylvain Hallé. 2018. Monitoring of Security Properties Using BeepBeep. In *Foundations and Practice of Security*, Abdessamad Imine, José M. Fernandez, Jean-Yves Marion, Luigi Logrippo, and Joaquin Garcia-Alfaro (Eds.). Springer International Publishing, Cham, 160–169.

[12] Robert Brayton and Alan Mishchenko. 2010. ABC: An Academic Industrial-Strength Verification Tool. In *Computer Aided Verification*, Tayssir Touili, Byron Cook, and Paul Jackson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 24–40.

[13] Fraser Cameron, Georgios Fainekos, David M. Maahs, and Sriram Sankaranarayanan. 2015. Towards a Verified Artificial Pancreas: Challenges and Solutions for Runtime Verification. In *Runtime Verification*. Springer International Publishing, 3–17.

[14] Ran Canetti, Pratik Sarkar, and Xiao Wang. 2020. Blazing Fast OT for Three-Round UC OT Extension. In *Public-Key Cryptography – PKC*. Springer International Publishing, 299–327.

[15] Ran Canetti and Mayank Varia. 2011. *Decisional Diffie–Hellman Problem*. Springer US, Boston, MA, 316–319. doi:10.1007/978-1-4419-5906-5_443

[16] David Carrel and Dan Harkins. 1998. The Internet Key Exchange (IKE). RFC 2409. doi:10.17487/RFC2409

[17] S Michael Crawford. 2017. Goodhart's law: when waiting times became a target, they stopped being a good measure. *BMJ* 359 (2017). doi:10.1136/bmj.j5425 arXiv:https://www.bmj.com/content/359/bmj.j5425.full.pdf

[18] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to play ANY mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC '87)*. Association for Computing Machinery, New York, NY, USA, 218–229. doi:10.1145/28395.28420

[19] Radu Grigore, Dino Distefano, Rasmus Lerchedahl Petersen, and Nikos Tzevelekos. 2013. Runtime Verification Based on Register Automata. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of ETAPS (Lecture Notes in Computer Science, Vol. 7795)*, Nir Piterman and Scott A. Smolka (Eds.). Springer, 260–276. doi:10.1007/978-3-642-36742-7_19

[20] Klaus Havelund, Doron Peled, and Dogan Ulus. 2018. DejaVu: A Monitoring Tool for First-Order Temporal Logic. In *2018 IEEE Workshop on Monitoring and Testing of Cyber-Physical Systems (MT-CPS)*. 12–13. doi:10.1109/MT-CPS.2018.00013

[21] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. 2011. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Conference on Security* (San Francisco, CA) *(SEC'11)*. USENIX Association, USA, 35.

[22] Mid Staffordshire NHS Foundation Trust Public Inquiry and R. Francis. 2013. *Report of the Mid Staffordshire NHS Foundation Trust Public Inquiry: Executive Summary*. Stationery Office. https://books.google.at/books?id=5K89wKY3RoMC

[23] Yu Jiang, Han Liu, Hui Kong, Rui Wang, Mohammad Hosseini, Jiaguang Sun, and Lui Sha. 2016. Use runtime verification to improve the quality of medical care practice. In *Proceedings of the 38th International Conference on Software Engineering Companion* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 112–121. doi:10.1145/2889160.2889233

[24] Samuel Judson, Ning Luo, Timos Antonopoulos, and Ruzica Piskac. 2020. Privacy Preserving CTL Model Checking through Oblivious Graph Algorithms. In *WPES'20: Proceedings of the 19th Workshop on Privacy in the Electronic Society,*

Virtual Event, USA, November 9, 2020, Jay Ligatti, Xinming Ou, Wouter Lueks, and Paul Syverson (Eds.). ACM, 101–115. doi:10.1145/3411497.3420212

[25] Jonathan Katz and Lior Malka. 2011. Constant-Round Private Function Evaluation with Linear Complexity. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security (Lecture Notes in Computer Science, Vol. 7073)*. Springer, 556–571. doi:10.1007/978-3-642-25385-0_30

[26] Joe Kilian. 1988. Founding crytpography on oblivious transfer. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (Chicago, Illinois, USA) *(STOC '88)*. Association for Computing Machinery, New York, NY, USA, 20–31. doi:10.1145/62212.62215

[27] Moonjoo Kim, Sampath Kannan, Insup Lee, Oleg Sokolsky, and Mahesh Viswanathan. 2002. Computational Analysis of Run-time Monitoring: Fundamentals of Java-MaC1. *Electronic Notes in Theoretical Computer Science* 70, 4 (2002), 80–94. doi:10.1016/S1571-0661(04)80578-4 RV'02, Runtime Verification 2002 (FLoC Satellite Event).

[28] Mika Kojo and Tero Kivinen. 2003. More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). RFC 3526. doi:10.17487/RFC3526

[29] John Kolesar, Shan Ali, Timos Antonopoulos, and Ruzica Piskac. 2025. Coinductive Proofs of Regular Expression Equivalence in Zero Knowledge. arXiv:2504.01198 [cs.CR] https://arxiv.org/abs/2504.01198

[30] Martin Leucker and Christian Schallhart. 2009. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming* 78, 5 (2009), 293–303. doi:10.1016/j.jlap.2008.08.004 The 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS'07).

[31] Yi Liu, Qi Wang, and Siu-Ming Yiu. 2022. Making Private Function Evaluation Safer, Faster, and Simpler. In *Public-Key Cryptography – PKC 2022*, Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe (Eds.). Springer International Publishing, Cham, 349–378.

[32] Ning Luo, Timos Antonopoulos, William R. Harris, Ruzica Piskac, Eran Tromer, and Xiao Wang. 2022. Proving UNSAT in Zero Knowledge. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS*. ACM, 2203–2217. doi:10.1145/3548606.3559373

[33] Ning Luo, Samuel Judson, Timos Antonopoulos, Ruzica Piskac, and Xiao Wang. 2022. ppSAT: Towards Two-Party Private SAT Solving. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 2983–3000. https:

//www.usenix.org/conference/usenixsecurity22/presentation/luo

[34] Ning Luo, Chenkai Weng, Jaspal Singh, Gefei Tan, Mariana Raykova, and Ruzica Piskac. 2024. Privacy-Preserving Regular Expression Matching Using TNFA. In *Computer Security - ESORICS 2024 - 29th European Symposium on Research in Computer Security (Lecture Notes in Computer Science, Vol. 14983)*. Springer, 225–246. doi:10.1007/978-3-031-70890-9_12

[35] Oded Maler and Dejan Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 152–166.

[36] Alex Mears. 2014. Gaming and Targets in the English NHS. *Universal Journal of Management* 2 (09 2014), 293–301. doi:10.13189/ujm.2014.020705

[37] Payman Mohassel and Seyed Saeed Sadeghian. 2013. How to Hide Circuits in MPC an Efficient Framework for Private Function Evaluation. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2013 (Lecture Notes in Computer Science, Vol. 7881)*. Springer, 557–574. doi:10.1007/978-3-642-38348-9_33

[38] Moni Naor and Omer Reingold. 2004. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM* 51, 2 (mar 2004), 231–262. doi:10.1145/972639.972643

[39] Amir Pnueli. 1977. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*. IEEE Computer Society, 46–57. doi:10.1109/SFCS.1977.32

[40] M. Rabin. 1981. *How to Exchange Secrets by Oblivious Transfer*. Technical Report. Technical Report Tech. Memo TR-81, Aiken Computation Laboratory.

[41] Masaki Waga, Kotaro Matsuoka, Takashi Suwa, Naoki Matsumoto, Ryotaro Banno, Song Bian, and Kohei Suenaga. 2024. Oblivious Monitoring for Discrete-Time STL via Fully Homomorphic Encryption. In *Runtime Verification - 24th International Conference, RV (Lecture Notes in Computer Science, Vol. 15191)*. Springer, 59–69. doi:10.1007/978-3-031-74234-7_4

[42] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 160–164. doi:10.1109/SFCS.1982.38

[43] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (SFCS 1986)*. 162–167. doi:10.1109/SFCS.1986.25

[44] Samee Zahur, Mike Rosulek, and David Evans. 2015. Two Halves Make a Whole. In *Advances in Cryptology - EUROCRYPT 2015*. Springer Berlin Heidelberg, Berlin, Heidelberg, 220–250.