A Formal Approach to Accountability in Heterogeneous Systems-on-Chip

Rajshekar Kalayappan, Smruti R. Sarangi

***Abstract*—Systems-on-chip (SoCs) are increasingly being composed of designs provided by different organizations. When such an SoC miscomputes or performs below expectation in-field, it is unclear which of the on-chip components caused the failure. The customer would like to use SoCs that provide the property of accountability, wherein the failure-causing component, and consequently its designing organization, can be unambiguously detected. Since it is a matter of trust, the various parties involved desire formal guarantees regarding any accountability solution. The solution must find the guilty component(s) in the event of a chip failure. Additionally, the solution must not falsely implicate any component that functioned correctly. This paper formally describes the property of accountability, a formal methodology of constructing an accountability solution, and a formal game-theory based methodology to reason about and prove the viability of a proposed solution. We explore the entire space of solutions, and characterize the attack surface and methods to provide accountability for each setting. We show non-intuitive results in this paper where seemingly simple solutions actually provide very powerful theoretical guarantees in terms of accountability.**

***Index Terms*—systems-on-chip, accountability, in-field fault localization, integration, game-theory, auditing**

I. Introduction

Systems on Chip (SoCs) are rarely designed entirely by a single organization. For example, many Qualcomm SoCs that go into most of our smartphones use general purpose compute cores designed by ARM. Companies such as Xilinx, Altera and Microsemi have licensed security microprocessor designs from Athena Group Inc [1]. Invia is another corporation that provides designs in the security co-processor domain [2]. Synopsys offers cryptographic accelerator IPs [3], and Intel too recognizes the fabless ASIC business and has thus started the Intel Custom Foundry [4] program. This is now a common practice in the semiconductor industry. Rather than having large monolithic organizations that design every part of the chip, the design effort is split across many organizations. This allows each individual organization to focus on particular aspects of the SoC, and develop world class expertise in that area. We call such SoCs, composed of designs from various organizations as *heterogeneous SoCs*.

An organization termed the System Integrator (SI) designs an SoC based on the requirements of the customer. The SI uses some component designs of its own, while it sources others from organizations termed as third party intellectual property vendors (3PIP vendors). The 3PIP vendors themselves commonly compose their designs using IPs from other 3PIP vendors [5]. For example, a JPEG encoder can use a DCT circuit, which has been designed by somebody else. This JPEG encoder can be a part of a large image processing chip. Thus, an SoC has a *fractal* design – at each level, an entity which we denote by the term 'host', composes a design using IPs from other entities whom we call 'guests'. A guest, in turn, can be a host at a lower level. The final design, containing designs from different organizations, is then fabricated. This model of *IP Reuse* is highly advantageous – organizations do not need to "re-invent the wheel", and it also allows an organization to focus on particular modules and achieve rapid progress in their designs. As examples, the Athena and Invia corporations have achieved expertise in the design of security co-processors. This expertise then translates to benefits for the entire semiconductor industry.

Despite the numerous advantages, there are a number of problems that arise from this approach as well. Since the guest is a different organization, the host (at each level) must be able to trust it, and the quality of its designs before incorporating them in its chip. Efforts are continuously being made to ease the process of *IP Qualification* [6][7] – to be able to quantify the quality of third party IP designs. This heterogeneous approach also introduces a new class of bugs – bugs arising at the interfaces between IPs from different organizations. These typically arise due to unclear communication of requirements from the host vendor to the guest vendor, and vice versa. Gajski et al. [8] recognized this issue from the early days of IP reuse, and researchers continue to develop new methodologies to reduce the occurrence of such bugs [9][10]. Benign bugs aside, the possibility of malicious bugs is also very real – Hardware Trojans are increasingly being discussed by both industry and academia. There are numerous real life examples that have altered geo-political scenarios [11]. Researchers are thus continually devising methods to detect the presence of malicious circuitry in third party IPs [12][13].

Increasing complexities of chips, and aggressive time-to-market deadlines, are reducing the time spent on Post-Si validation. This increases the chances of a bug slipping to the field. Now should a chip fail on field, who is to be held accountable, and compensate the customer whose application suffered? The chip is a result of the design efforts of multiple organizations.

Another aspect of the problem comes about from economic underpinnings. The different vendors are business organizations in a competitive market. Should an SoC malfunction on field, a vendor – host or guest – who has contributed towards

Rajshekar Kalayappan is with the Department of Computer Science and Engineering, Indian Institute of Technology, Dharwad, India e-mail: rajshekar.k@iitdh.ac.in

Smruti R. Sarangi is with the Department of Computer Science and Engineering, Indian Institute of Technology, Delhi, India e-mail: srsarangi@cse.iitd.ac.in

part of the design, does not want to be held responsible for the malfunction. This will detrimentally effect her position in the market. Therefore, it is rational for a vendor to shift blame for her own shortcoming to someone else, should it be possible to do so. Thus, the intended solution to this already complex problem of debugging an SoC, should also have to account for vendors potentially compromising the debug process to protect their own interests.

All the mentioned issues with IP reuse can be captured by the notion of *accountability*. Any party, be it a host or a guest, must be accountable for any bugs found in its design. This notion of accountability was introduced by Kalayappan et al. [14]. Existing debug hardware falls short of providing accountability because it does not account for the possibility of the host or the guest components possibly compromising the debug process to escape being held accountable. Existing debug hardware assumes the sanctity of all logs – that they are an accurate reflection of what transpired in the field. However, a rational vendor (host or guest) will compromise the debug process to escape being held responsible for a chip malfunction, should it be possible to do so. This brings about the need for a more sophisticated approach to debugging, one where the sanctity of the logs can be guaranteed in the presence of malicious on-chip components.

Kalayappan et al. [14] presented a practical implementation of a solution that provided accountability in a heterogeneous SoC, with minimal area, power and performance overheads. The viability of the approach was also demonstrated using bugs from the errata documents of real SoCs. The solution makes use of embedded "meters" in the chip. These meters are designed by Trusted Third Party vendors – vendors who are trusted by all vendors (e.g., Trusted Computing Group [15], SiidTech [16]). These meters help produce authentic logs of events that transpire in the field. These logs may then be analyzed should the chip malfunction, in order to determine the responsible component.

With viable accountability solutions now practical, a formal treatment of the design of an accountability solution is the need of the hour. Like security, accountability too is a property whose solutions require formal proofs to convince the parties to adopt them. We must be able to provide guarantees that no bug slips through uncaught. We must also be able to provide guarantees that only the vendor with the faulty design is held accountable. An innocent vendor must not be wrongly implicated by our system. In this paper, we work towards a formal definition of an accountable chip, and a formal framework for designing and reasoning about the validity of an accountability solution. We model the scenarios and solutions as formal games and solve the games to comment on their viability. Such a game-theory based approach for validation is commonly adopted when dealing with notions of trust and security in systems [17][18][19].

We first discuss in Section II some related work from other domains of chip design that bear some similarities in spirit with accountability. We then, in Section III, informally introduce the reader to the accountability problem and the general solution approach that we propose. We follow this with a formal treatment of the property of accountability in Section IV, and proceed to prove how accountability can be achieved through audited logging of messages exchanged across different domains on the chip in Section V. We present multiple ways to achieve the required audited message log, and compose a library of these *auditing primitives* in Section VI. We then apply the different primitives to a range of scenarios in Section VII, where a scenario is an ⟨SoC model, attack model⟩ tuple. Each such application of a primitive is an accountability solution. We present a formal game-theory based approach to reason about an accountability solution. This is important in matters involving trust (such as security and accountability) because a formal reasoning methodology is essential to instill confidence in the stakeholders to adopt the solution. The formal approach not only helped us prove that one of our proposed solutions works in all recognized scenarios, it also helped us prove the viability of an interesting, intuitively *unfair*, design. This valid design can achieve qualified accountability with a significant reduction in overheads.

II. Related Work

A. Reliability and Hardware Security

Research in the reliability and hardware security domains also deals with identifying the occurrence of unsatisfactory behavior in-field (please refer to [20] for a comprehensive survey of techniques). Typical classes of solutions are based on redundancy, invariant verification, and symptom detection.

In the case of redundancy based solutions, we assume that the components suffer only benign faults. This allows redundancy to be able to detect a naturally occurring phenomenon such as a permanent fault or a transient single event upset [21], [22]. Alternatively, we assume that the components are sourced from different vendors (designers / fabricators). This is applicable to security solutions [23], [24]. If it can be assumed that the vendors will not collude, then such a redundant arrangement helps detect a malfunction. In the case of accountability however, neither are the malfunctions benign, nor can anything be assumed regarding the vendors of the different components.

In the case of invariant verification, and symptom detection, the test units (popularly called Built-In Self Test units, or BIST units) are typically designed by the host itself [25]. Adopting this approach to achieve accountability is unfair to the guest vendors. The host may very well incriminate a guest component for failures of its own. The aim of this work is to create a framework that provides a solution to this problem.

B. Logging-based Bug Isolation Solutions

Many bugs escape pre-silicon and post-silicon validation to reach the field [25]. Researchers have proposed employing logging to record the functioning of the chip, and analyzing the logs, either online or offline, to isolate the location of the bug [26], [27].

Kalayappan et al. [28] present a scheme to log the interactions between the host circuitry and third party accelerators. They consider a scenario where the host and the accelerator vendors do not trust each other, and some form of on-chip, trusted third party auditing is required. The reliable

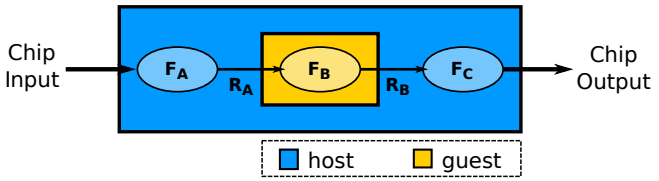


Fig. 1: An informal illustration of Accountability

logs thus collected can be used for a variety of purposes including debugging, analyzing performance bottlenecks, and investigating violations in security. The solution however is for a very limited scenario, and unlike this paper, does not investigate the space of all possible attack scenarios.

C. Malicious Contractors in Outsourced Computation

The problem at hand, when abstracted, is highly similar to that of malicious contractors in the cloud domain [17]. Consider a cloud composed of different contractors who lease out their compute resources in return for remuneration. A user of this cloud submits a compute task to it. She is not concerned about which contractor does the job, as long as the job is done correctly. However, a contractor, in the interest of maximizing profits, may maliciously *"under-compute"* and return an approximate result. This scenario is very similar to our problem – the cloud is analogous to the SoC, while the malicious contractor is analogous to the third party IP.

The solution proposed by Kupcu [17] to counter this problem of malicious contractors is to have multiple contractors redundantly compute the job and compare their results. The assumption is that a majority of the contractors are honest. However, having redundant implementations of a module on chip is a massive burden on chip area. Therefore, we must look at a different approach.

This domain of research proves results by formulating scenarios as games – something we also do in this paper.

III. An Informal Introduction to the Problem of Accountability and the Proposed Solution Approach

Let us consider a simple illustrative example. As shown in Figure 1, the chip consists of some circuitry designed by the host, called the host circuitry, and a single third party IP, known as the guest circuitry. Let us suppose that the chip functions in the following way: the end-user submits an input to the chip. This is received by the host. The host performs some function F_A on the input, and the result is R_A. The host gives R_A to the guest who performs some function F_B on it, to produce R_B. The guest gives R_B to the host, who performs some function F_C on it, to produce R_C. This R_C is then returned to the user as the final result.

Now if the chip has malfunctioned, the proposal is to analyze the intermediate results – the messages passed between components belonging to different vendors – to identify the responsible component. In this case, R_A and R_B have to be analyzed – if R_A was correct, and R_B was incorrect, then the guest is responsible. If this is not the case, then the host is responsible.

The proposal is that R_A and R_B are logged by the host, and made available for offline analysis when required. Now it is possible that the host or the guest tampers with the logged R_A or R_B thereby escaping responsibility. To counter this, trusted, tamper-proof meters are embedded strategically that provide certificates for the logs. The certificates certify the content of the message passed between two components designed by different vendors, as well as the time of transfer. It is the host's responsibility to store the certificates along with the logs. In the event of the chip malfunctioning in the field, the stored certified logs are analyzed offline to determine if it was the host or the guest that was responsible.

The certificates are essentially cryptographic hashes of the message contents and the time of transfer. The hashes are produced using secret keys known only to the meters. Therefore, neither the host nor the guest can tamper a log and produce a fake certificate for it. Neither can they maliciously delete a log of a transferred message (techniques such round-based cryptosystems may be employed [14]), nor add to the logs a message that was never transferred. How these meters are placed, and how the authenticity of the logs are *formally guaranteed*, form the contributions of this work.

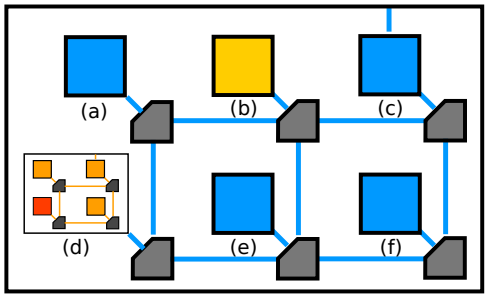
For rigorous implementation details and practical considerations, please see Section VIII and the work by Kalayappan et al. [14].

IV. Formally Defining Accountability

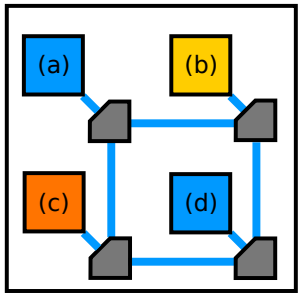
We first describe some formalisms related to the notion of accountability that were introduced by Kalayappan et al. [14]. We present a more thorough treatment here as these formalisms are required to describe our contributions.

A. Model of a Heterogeneous SoC

Figure 2(a) describes a generic model of a heterogeneous SoC containing IPs from many different vendors. The convention followed is that components designed by the same vendor are marked by the same color. As discussed in the introduction (Section I), the design of a modern SoC has a *fractal* nature. At each level, an entity named "host" composes a design using some of its own IPs, and sourcing others from "guest" entities. In Figure 2(a), at the topmost-level, the host is the blue entity, while the yellow, orange, and gray entities are guests. If we consider a lower level, component (d) has the orange entity as the host and the red and dark gray entities as the guests. At each level, let the host circuitry be denoted by H, and the set of all guest circuitries be denoted by $\mathbb{G}$. Note that the host of the topmost level is the System Integrator (SI). Also note that the guests cannot communicate with each other directly. Any communication between two guests has to pass through the host. Specifically, we focus on network-on-chip (NoC) based SoCs in this work. The NoC, which is responsible for providing the network interface to each component as well as routing packets towards their destination, may be part of the host circuitry or may be externally procured, that is, a guest.



(a) Generic SoC Model



(b) Simplified Representative SoC Model

Fig. 2: Model of a 3PIP-containing SoC

In Figure 2, the NoC, depicted in gray (unlabeled routers), is a guest.

When the host decides to incorporate a particular guest G, it enters into an agreement regarding the conditions of its use with the guest vendor. First, the nature of the inputs to the guest, $IpCond_G$, are decided upon. For example, in the case of a JPEG accelerator, the possible dimensions of the input image may be agreed upon by the SI and the guest vendor. Second, the nature of the output of the guest, $OpCond_G$, is negotiated. This could be the actual result of a computation, as is the case in, say, a cryptographic accelerator. It could also be a property of the result. For example, in the case of a compression accelerator, the desired compression ratio may be agreed upon. Third, the quality of service (QoS) – that is, the guest's latency/throughput while performing a job, $QoSCond_G$ – is agreed upon. Fourth, the quality of the environment (QoE) in which the guest operates, $QoECond_G$, is agreed upon. QoE refers to the latency/throughput provided by H for servicing resource requests (e.g., last-level cache requests) made by G.

For the purpose of accountability, it is sufficient to consider each level separately. For instance, let us suppose the chip in Figure 2(a) malfunctioned in the field. Further, suppose the first round of analysis at the topmost level, with blue as host and yellow, orange, and gray as guests, held orange (component (d)) accountable. The second round of analysis focuses on component (d) – with orange as the host and red and dark gray as the guests – and proceeds in the same fashion as done for the first level, and could potentially hold orange, dark gray, or red accountable. Since the analysis at each level proceeds in the same fashion, it is sufficient to design an accountability solution for a single level. The accountability analysis for a fractal SoC also proceeds in a fractal fashion. This leads us to consider a simplified representative model of an SoC, as shown in Figure 2(b), for all further discussion in this paper. Here, blue is the host, and yellow, orange and gray are guests.

B. Trust Model

Any host or guest vendor (at any level): (i) may malfunction because of benign design bugs, (ii) may malfunction because of malicious Hardware Trojans, (iii) is rational, meaning, it will protect its own interests and deflect the blame for its own shortcomings to another vendor, should it be possible to do so.

A trusted third party (TTP) vendor is present, who is trusted by all other host and guest vendors. The TTP provides light-weight auditing meters (see Section VI) that the hosts and guests embed in their designs. The meters contain cryptographic circuitry [14] and associated embedded keys [29]. The meter designs are tamper-proof [30]. The foundries are assumed to be trusted [31], [32].

Though our solutions assume only the TTPs to be trusted, a more realistic scenario is to have the System Integrator – that is, the host at the topmost level – as a trustworthy entity as well. This allows the SI to function as the TTP, and to provide meter designs to the other hosts and guests on the chip. Thus, accountability at all the lower levels other than the topmost level can be provided through the proposed solutions.

C. The Problem of Accountability

The user employs the SoC to perform *tasks*, denoted by the set $\mathbb{T}$. In performing a task $t \in \mathbb{T}$, H carries out some computations of its own, and can request the services of the on-chip guests for others. We term requests made by the host to the guests as *jobs*. The jobs corresponding to a task t are denoted by the set $\mathbb{J}_t$. Let the input provided to some guest G as part of some job $j \in \mathbb{J}_t$ be Ip_j, the output be Op_j, the quality of service provided be QoS_j, and the quality of environment be QoE_j.

Now it is possible that the result returned to the user is erroneous. Let the function $FF(t)$ denote that the execution of task t faced a functional failure. It is also possible that task t took too long to complete. Let the function $TF(t)$ denote that the execution of task t faced a timing failure. Both return Boolean values.

Definition 1: An `accountable` heterogeneous SoC is defined as one in which the cause of the error (functional failure) or the delay (timing failure) can be unambiguously attributed to the host and/or one or more of the guests.

We next define the rules governing the assignment of responsibility for an error to either H or any $G \in \mathbb{G}$. Let the function $R(x)$ denote that module x was responsible for the error, with $x \in \{H, \mathbb{G}\}$.

The function $Xsat(j, x)$ is used to denote whether the measurement of X, when module x performed job j, satisfied the agreed upon conditions, with $x \in \mathbb{G}$, $j \in \mathbb{J}_t$, and t being the task whose execution requires analysis. For example, $QoSsat(j, G)$ denotes whether the measured QoS when

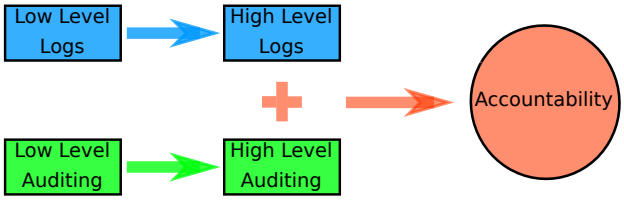


Fig. 3: A logging-based solution to achieve accountability

module G executed job j satisfied the agreed upon condition $QoSCond_G$.

Definition 2: The assignment of responsibility for an erroneous execution of a user's task t is done according to Axioms 1, 2, 3, and 4.

Axiom 1: $FF(t) \wedge \exists j \in \mathbb{J}_t(\neg OpSat(j,G) \wedge IpSat(j,G)) \rightarrow R(G)$

If the dispute is regarding an incorrect task result, and if there is a guest G such that it received satisfactory inputs but produced unsatisfactory outputs, then G must be held responsible.

Axiom 2: $FF(t) \wedge \forall j \in \mathbb{J}_t(\neg IpSat(j,G) \vee OpSat(j,G)) \rightarrow R(H)$

If the dispute is regarding an incorrect task result, and if there is no guest G such that it received satisfactory inputs but produced unsatisfactory outputs, then H must be held responsible.

Axiom 3: $TF(t) \wedge \exists j \in \mathbb{J}_t(\neg QoSSat(j,G) \wedge QoESat(j,G)) \rightarrow R(G)$

If the dispute is regarding a task result taking too long to be computed, and if there is a guest G such that it observed satisfactory QoE but produced unsatisfactory QoS, then G must be held responsible.

Axiom 4: $TF(t) \wedge \forall j \in \mathbb{J}_t(\neg QoESat(j,G) \vee QoSSat(j,G)) \rightarrow R(H)$

If the dispute is regarding a task result taking too long to be computed, and if there is no guest G such that it observed satisfactory QoE but produced unsatisfactory QoS, then H must be held responsible.

V. Accountability through Logging

Guided by the definition of accountability described in Section IV, we propose to log the functioning of the different components *in-field*. We will capture the Ip_j, Op_j, QoS_j, and QoE_j for each job j performed by any guest component. When the chip malfunctions, and the customer wishes to know which on-chip component is responsible, the collected logs are analyzed offline (note that collecting all logs may pose a large performance and storage penalty. We address these concerns in Section VIII).

The overview of the idea is given in Figure 3. We will now discuss each component in detail.

A. High Level Logging

Definition 3: High Level Logging refers to the logging of Ip_j, Op_j, QoS_j, and QoE_j for each job j executed by a guest G, $\forall G \in \mathbb{G}$: (i) $IpLog_j$: consists of all communication from H to G, (ii) $OpLog_j$: consists of all communication from G to H, (iii) $QoSLog_j$: recording of the QoS provided, (iv) $QoELog_j$: recording of the QoE provided. These four logs constitute the *high level logs*.

The host places circuitry to log the high level logs at each of its interfaces with guests, and its interface with the external world. Existing design-for-debug (DfD) infrastructure such as trace buffers, which are idle in-field, may be reused for this purpose. Alternatively, dedicated logging structures may be commissioned. The collected logs are periodically written off-chip to disks owned by the customer, again, just as done in post-silicon validation. The customer, when faced with a chip malfunction, presents the logs for offline analysis.

B. High Level Auditing

As discussed in Section I, the various organizations – hosts, guests, and the customer – do not necessarily trust each other. Let us consider the worst case. If any entity can profit from any misbehavior, it will misbehave. This defines the rationality of each organization. If the host or the guest vendor can escape being held responsible for a chip malfunction by performing some sort of misbehavior, it will do so. If the customer can force a chip malfunction and get compensated for it, it will do so.

Our accountability solution must accurately determine if there was a chip malfunction in-field, and also accurately determine which of the on-chip components was responsible. It must be able to counter any misbehavior attempts made by these rational players.

The accurate provision of accountability rests on the accuracy of the collected logs. If any organization compromises the logs, then the solution fails. We propose to employ an on-chip auditing system that certifies the logs. The certificates are stored along with the logs. Only those logs accompanied by certificates are authentic, and are considered for the offline analysis.

Definition 4: The high level auditing system provides four certificates for each job j:
$IpCert_j$ certifying $IpLog_j = Ip_j$,
$OpCert_j$ certifying $OpLog_j = Op_j$,
$QoSCert_j$ certifying $QoSLog_j = QoS_j$,
$QoECert_j$ certifying $QoELog_j = QoE_j$.

Theorem 1: A heterogeneous SoC is `accountable` if and only if the four logs – $IpLog_j$, $OpLog_j$, $QoSLog_j$, and $QoELog_j$, and the four certificates to authenticate the collected logs – $IpCert_j$, $OpCert_j$, $QoSCert_j$, and $QoECert_j$, are available, $\forall j$ jobs performed by any guest $G \in \mathbb{G}$.

Proof: By Definition 2, Ip_j, Op_j, QoS_j, and QoE_j, for each job j performed by any $G \in \mathbb{G}$, is necessary and sufficient to find out the responsible on-chip component(s) and provide accountability.

By Definition 4, the four certificates ensure that the four logs collected by the host – $IpLog_j$, $OpLog_j$, $QoSLog_j$, and $QoELog_j$ – correspond to what actually transpired during the in-field execution, that is – Ip_j, Op_j, QoS_j, and QoE_j.

Thus, the four logs, accompanied by the four certificates, are necessary and sufficient to provide accountability.

C. Low Level Logging

Practically collecting the four high level logs – Ip, Op, QoS, and QoE – with minimal overhead is difficult. We propose to collect instead "low level logs".

Definition 5: Low level logs capture each message communicated, between the host and a guest (recall that two guests cannot directly communicate with each other). For each message that is communicated, (i) the contents of the message, (ii) the sender and receiver IDs, and (iii) the time of transfer, are logged.

Now we prove how low level logs are sufficient to derive the high level logs offline.

Lemma 1: $IpLog_j$, for each job j performed by any guest G, can be derived from low level logs.
Proof: This follows directly from Definition 3 that states that $IpLog_j$ is the log of all communication from H to G as part of job j. Aggregating the contents of all messages from H to G therefore gives $IpLog_j$.

Lemma 2: $OpLog_j$, for each job j performed by any guest G, can be derived from low level logs.
Proof: This follows directly from Definition 3 that states that $OpLog_j$ is the log of all communication from G to H as part of job j. Aggregating the contents of all messages from G to H therefore gives $OpLog_j$.

Lemma 3: $QoSLog_j$, for each job j performed by any guest G, can be derived from low level logs.
Proof: This follows directly from Definition 3. The QoS provided by G is determined by the time when it received the job description, and the time when it provided the result to H. The latency can therefore be derived from the times of transfer of the job description message from H to G, and the result message from G to H. If the result of G is a stream of messages, the throughput can be derived from the times of transfer of the constituent messages.

Lemma 4: $QoELog_j$, for each job j performed by any guest G, can be derived from low level logs.
Proof: This follows directly from Definition 3. The QoE provided to G is determined by the times when it made a resource request, and the times when it received the responses. The latency can therefore be derived from the times of transfer of the resource request messages from G to H, and the response messages from H to G. If the input to G is a stream of messages, the throughput can be derived from the times of transfer of the constituent messages.

Theorem 2: The high level logs can be derived offline by aggregating low level logs.
Proof: The proof follows directly from Lemmas 1, 2, 3, and 4.

D. Low Level Auditing

Just like in the case of high level logs, the economical realization of high level certificates is also difficult. Instead, we propose to perform certification at the granularity of messages transferred from the host to the guest, or vice versa. That is, we wish to perform certification at the granularity of low level logs. (Note that this is an abstraction necessary to discuss the formal underpinings of providing accountability. Practically, logging every exchanged message may prove intractable – please see Section VIII for a discussion on practical considerations).

Since the exact same certification has to be done for both messages from the host to the guest, and vice versa, we can view the problem as providing certification in an abstract sender-receiver paradigm. The sender (receiver) can be either the host (guest) or the guest (host). The requirements of such a low level auditing system is formally defined in Section VI, followed by an intuitive construction of a set of low level auditing solution or *primitives*. Section VII then presents a taxonomy of SoCs. For each SoC class, we apply the different primitives to perform low level auditing. We also vary the attack model that tries to sabotage the audit. The different combinations result in systems that achieve high level auditing, and consequently accountability, to different degrees.

VI. Auditing Messages in a Sender-Receiver Paradigm

A. The Problem

Consider two mutually distrusting nodes S and R connected by a single link that is assumed to be non-faulty. S (the sender) performs some computation, and sends the result as a message to R (the receiver). The message serves as an input to R, who begins computing after receiving the message. Let us consider a single message transmission. Let the sender send message MSG_S at time T_S. Let the receive get the message MSG_R at time T_R. If the receiver did not get any message, then $MSG_R = T_R = \phi$, and if the sender did not send any message, then $MSG_S = T_S = \phi$.

The auditing system has to produce a certificate $\mathbb{C}$, that consists of (i) S_C, the certified sender, (ii) R_C, the certified receiver, (iii) MSG_C, the certified message contents, and (iv) T_C, the certified time of communication.

Definition 6: For the auditing system to be `sound`, it must guarantee the following four properties:

1) **Non-Repudiation**: $S_C = S$ AND $R_C = R \Rightarrow$ The actual sender and receiver are certified.
2) **Integrity**: $MSG_S = MSG_R = MSG_C \Rightarrow$ The message was received and certified as sent.
3) **Timeliness**: $|T_S - T_C| < \tau$ AND $|T_R - T_C| < \tau$, where τ is a predefined (small) positive constant $\Rightarrow$ The actual (within a small margin of error) time of transfer is certified, and no undue delay is induced between the sending and receipt.
4) **Atomicity**: Either $(((MSG_S = MSG_R = MSG_C) \neq \phi) \wedge ((T_S \approx T_R \approx T_C) \neq \phi))$ OR $((MSG_S = MSG_R = MSG_C = \phi) \wedge (T_S = T_R = T_C = \phi))$. The message is either sent and received, and certified and logged (and the corresponding times are approximately the same) or not sent/received/certified and logged at all.

We derive these properties from non-repudiation research [33] that essentially deals with providing certificates to the sender

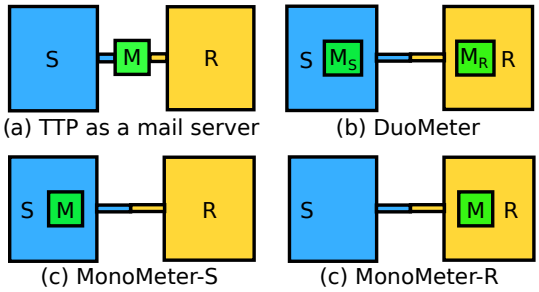


Fig. 4: Candidate solution: using a trusted third party ("M" in the figures)

and receiver of a message that allow them to prove to an adjudicator that they sent and received the message respectively.

The rest of this section will focus on compiling a library of primitives – solutions to the above defined problem. Section VII will apply the different primitives to form the low level auditing system in a heterogeneous SoC, and prove to what degree the four high level certificates, and consequently accountability, can be attained.

B. Solution 1: Trusted Third Party(TTP) providing a Mail Service

Assume a meter node, M, that is trusted by all parties (see Figure 4(a)). S sends the message MSG to M at time t, requesting it to forward it to R. Aside from forwarding the message, M produces a certificate $\mathbb{C}$ with $S_C = S$, $R_C = R$ (non-repudiation), $MSG_C = MSG$ (integrity), and $T_C = t$ (timeliness). $\mathbb{C}$ is essentially a cryptographic hash of (S_C, R_C, MSG_C, T_C), made using a secret key known only to M, the trusted third party. Thus, during an offline dispute, the TTP organization can verify if the certificates are genuine.

Additionally, M produces $\mathbb{C}$ for every message that is sent to it by S, and also forwards the message to R. It does not produce $\mathbb{C}$ for messages it did not receive. However, it is possible that $\mathbb{C}$ (and the corresponding log) is discarded by S or R while resolving a dispute offline. To counter this, we propose to employ a cryptographic system whose state (crypto-state) *changes with every round*, such as PRESENT [34]. In such systems, the cryptographic key changes in a certain predefined way after each round of encryption. Consequently, if the certificate of the k^{th} message, $\mathbb{C}^k$, is discarded during an offline dispute, this can be detected. It will not be possible for any of the parties to read the $(k+1)^{th}$ message or verify $\mathbb{C}^{k+1}$ without processing $\mathbb{C}^k$.

It is difficult to employ this solution in SoCs where all vendors distrust each other, because it requires M to have direct (and reliable) links to both the sender and the receiver. The TTP provides meter designs to the host and guest vendors in the form of hard IPs. The latter embed the meters in their designs, thereby, making it impossible for a meter to have direct links to multiple parties.

C. Solution 2: Solution using Redundant Embedded Meters "DuoMeter"

In the *DuoMeter* scheme, trusted third party (TTP) meters are embedded at both S and R (see Figure 4(b)). S sends the message MSG_S intended for R to the TTP meter (M_S) embedded in it at time T_S. M_S encrypts (S, R, MSG_S, T_S) using a secret key it shares with M_R (receiver side trusted meter), and sends the encrypted message to S. M_S also produces a certificate $\mathbb{C}_S$, a hash of (S, R, MSG_S, T_S) produced using the same secret key. S then sends the encrypted message to R, who forwards it to M_R at time T_R, claiming that it is a message from S. The latter decrypts the message MSG_R and sends it to R, who consumes it. M_S also produces a certificate $\mathbb{C}_R$, a hash of (S, R, MSG_R, T_R).

Non-repudiation: When M_R decrypts the message, it verifies that (i) the sender ID in the message matches the sender ID claimed by R, and (ii) the receiver ID in the message matches the ID of the node it is embedded in.

Integrity: The message sent from M_S to M_R (through S and R) is accompanied by a *checksum*, which is essentially $\mathbb{C}_S$. If S or R tamper with the message after it is certified by M_S, the verification of the checksum at M_R will fail. However, it is not possible to identify whether S or R or both modified the message. This is similar to the *last-link* problem encountered in accountability solutions for the Internet [35]. We term this attack as an *irrational integrity attack*. We use the word "irrational" because S and R have no immediate gain from performing such an attack [17]. S cannot hide a wrong computation by it by modifying the message after the $\mathbb{C}_S$ was produced because $\mathbb{C}_S$ was produced on the message given by S to M_S. Similarly, modifying the message does not help R evade a wrong computation. R has no way of knowing at the time of communication (before it sends to M_R) whether the input will cause it to compute wrongly since the message is unreadable (encrypted). The only motivation for S and R to perform such an attack is to bring down the performance of the system as a whole through the penalties paid for recovering from the attack.

Timeliness: Let T_{CS} be the certified time of sending the message, while T_{AS} be the actual time of sending. Similarly, let T_{CR} be the certified time of receiving the message, while T_{AR} be the actual time of receiving. Now, a scenario where $(T_{CS} < T_{AS})$ is advantageous to S since this shortens the certified duration of S's computation. Similarly, a scenario where $(T_{CR} > T_{AR})$ is advantageous to R since this shortens the certified duration of R's computation.

In the DuoMeter scheme, since S cannot modify the message after it is certified by M_S, S is forced to complete all its work before the $\mathbb{C}_S$ is produced. Since the message is encrypted, it has to be decrypted by M_R before R can use the message. Thus R is forced to begin its work only after $\mathbb{C}_R$ is produced. Thus, neither S nor R can subvert the timeliness property to their advantage. However, they may still perform an *irrational timeliness attack* similar to the irrational integrity attack. The only effect of this will be to bring down the performance of the entire system. Now, since the message received at M_R contains the time of sending T_S as well, M_R can use this to check if any undue delay was induced by T_S, i.e., if $T_R - T_S > \tau$ (τ is a predefined constant). Thus, just like the *irrational integrity attack*, an *irrational timeliness attack* can be detected, but it is not possible to find out who had inserted the delay (S or R).

Atomicity: It is possible that S or R drops the message after the $\mathbb{C}_S$ is issued. This scenario can be detected by the round-based crypto-system described in Section VI-B. All further communication between S and R will fail if a message is dropped. Thus, an *irrational atomicity attack* can be detected. This attack is an *irrational* one as well – neither S nor R serve to gain anything from doing this. And just like the other two irrational attacks, the *irrational atomicity attack* can be detected but it cannot be ascertained who the guilty party is. An attack where both the log and message are discarded is also countered through employing a round-based crypto-system as described in Section VI-B.

Thus, the redundant metering solution is a `sound` auditing system. The identities of the sender and the receiver, the contents of the message, the time of sending and the time of receiving can be accurately certified. Atomicity is also guaranteed. All types of irrational attacks can be detected, but the guilty party cannot be ascertained.

The two certificates $\mathbb{C}_S$ and $\mathbb{C}_R$ are in a sense redundant since they certify the same features of the message transfer ($T_S \approx T_R$, if timeliness is not violated). However, having the certificates generated at both S and R makes the storing of the certificates along with the logs in SoCs easier.

1) Solution 3: Single Embedded Meter "MonoMeter": In this scheme, only a single embedded meter is employed. Let us define *MonoMeter-S*, where the embedded meter is in the sender S (Figure 4(c)). When S needs to send a message MSG_S to R, it sends it to the trusted meter M. Let the time be T_S. M produces C by hashing (S, R, MSG_S, T_S). S then sends the message to R.

MonoMeter-S is not a `sound` auditing system. $\mathbb{C}$ merely certifies that MSG_S was produced by S and ready to be sent to R at T_S, and nothing more. The message may be sent to a different recipient (violating non-repudiation), may be modified (violating integrity), may be delayed (violating timeliness), or may be dropped altogether (violating atomicity).

In *MonoMeter-R*, the embedded meter is in the receiver R (Figure 4(d)). When S needs to send a message MSG_S, it encrypts it using a secret key it shares with the trusted meter embedded (M) in R. It computes a checksum as well. It sends the encrypted message to R along with the checksum. R sends these to M at T_R, which decrypts it to produce MSG_R, verifies the checksum and produces $\mathbb{C}$ by hashing (S, R, MSG_R, T_R). M then sends MSG_R to R.

MonoMeter-R is not a `sound` auditing system. If the integrity check (checksum verification) at M succeeds, $\mathbb{C}$ merely certifies that MSG_R was produced by S and ready to be consumed by R at T_R (and not before), and nothing more. The message may be delayed (violating timeliness), or may be dropped altogether (violating atomicity). Non-repudiation is guaranteed since if the sender or receiver identities are not as claimed, the checksum verification, which is based on a secret key shared between S and M (embedded in R), will fail. Note that R cannot spoof messages or perform replay attacks if the cryptosystem is a round-based one as proposed. Similarly, the checksum verification also detects attacks on integrity.

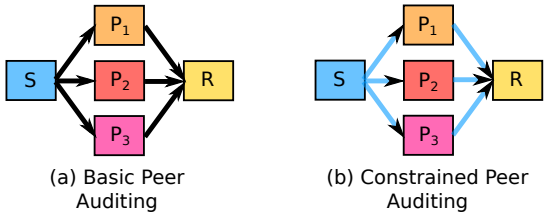


Fig. 5: Candidate solution: Peer auditing schemes

Despite not providing a `sound` audit, we will see that for certain attack models (see Sections VII-C2,VII-D2, and VII-E1), a single embedded meter can be quite useful.

D. Solution 4: Peer Based Auditing

A TTP-less auditing solution is possible by following a *peer auditing* approach. If an assumption can be made that no more than k peers are malicious, then having $2k+1$ peers providing certificates is sufficient for a sender (or receiver) to prove its claim.

1) BasicPeer*: Basic Peer Auditing:* In the basic peer auditing scheme, the links between nodes are trustworthy. Figure 5(a) illustrates this case. In this example, the maximum number of malicious intermediaries is assumed to be one. A malicious intermediary is one who may conspire with S or with R or be arbitrarily malicious.

S sends the message MSG_S, intended for R, to $2k+1$ (=3 in this example) intermediaries. Each of the intermediaries functions like the TTP meter in the mail server primitive (Section VI-B). The intermediaries forward the message to R as MSG_R , and also produce certificates. Under the k maximum adversary assumption, a majority of the certificates attest that $S_C = S$, $R_C = R$ (non-repudiation), and $MSG_C = MSG_S = MSG_R$ (integrity).

Regarding timeliness, during an offline dispute, S chooses $k+1$ certificates with the earliest T_S. Among these, the certificate with the latest T_S is chosen. Similarly, R chooses $k+1$ certificates with the latest T_R, and the earliest one among these is chosen. Such a protocol guarantees timeliness. S can be victimized by the k malicious intermediaries if the latter issue certificates that say the T_S was much later than what it actually was ($T_{CS} >> T_{AS}$). But since the $k+1$ benign intermediaries provide benign certificates, S is unaffected by any malice. On the other hand, S may choose to conspire with the k malicious intermediaries to achieve a T_S much earlier than what it actually was ($T_{CS} << T_{AS}$). But since the protocol utilizes the latest T_S among S's produced $k+1$ certificates, the k malicious certificates are rendered harmless. The timeliness guarantee of T_R can also be reasoned in a similar vein.

Atomicity is also guaranteed as the majority of the intermediaries are benign, and they always issue a certificate for every message they receive, and forward every message they certify. Hence, *BasicPeer* is a `sound` auditing scheme.

2) ConstrainedPeer*: Constrained Peer Auditing:* In 3PIP-containing SoCs, the trustworthiness of the links cannot be guaranteed. The links may be designed by S or R giving it the upper hand. Figure 5(b) illustrates the scenario when S

controls the links between entities. This situation is similar to having an embedded meter in S, that is, the *MonoMeter-S* scenario (Section VI-C1), under the k maximum adversary assumption. The trustworthy majority of the certificates attest that the certified message was produced by S and ready for sending to R at the certified time of sending, and nothing more. Similarly, if the links are controlled by R instead, then the situation is similar to having an embedded meter in R, that is, the *MonoMeter-R* scenario (Section VI-C1), under the k maximum adversary assumption.

This brings us to the end of the discussion on alternatives to auditing messages in a sender-receiver paradigm. Our library of auditing primitives thus consists of: **(1)** TTP-as-a-mail-server, **(2)** DuoMeter, **(3)** MonoMeter, **(4)** BasicPeer, **(5)** ConstrainedPeer. Let us now use these primitives as low level auditing systems in 3PIP containing SoCs.

VII. Accountability in 3PIP-containing SoCs

As discussed in Section IV, high-level logs are sufficient to provide accountability in SoCs if the logs can be authenticated by an auditing system. In this section, we will apply the low-level auditing primitives compiled in Section VI to different types of SoCs, vary the attack model of the auditing system's saboteur, and prove to what degree the high-level certificates – $IpCert$, $OpCert$, $QoSCert$, and $QoECert$ – can be generated.

A. Preliminaries

Classification of 3PIP-containing SoCs

Figure 6 depicts our proposed classification. The first class of SoCs are those with hosts trusted by the guests. The second class of SoCs are those where the host is untrusted by the guests, and the NoC is designed by the host. The third class of SoCs are those where the NoC is designed by a trusted guest, and the fourth class are those where the NoC is designed by an untrusted guest. Guests (other than the trusted NoC vendor in the third class) are untrusted by the host in all SoC classes. We present a range of solutions for each class of SoCs – it is up to the designers of the SoC to decide which solution is the most apt with respect to the particular scenario and budget.

Taxonomy of Attacks on the Auditing System

A party may choose to sabotage the auditing system for two reasons: (i) to evade from being implicated for its own functional or timing error, or (ii) to implicate another party, even though its own performance is satisfactory. We call the second type of attacks "business related irrational attacks" (BRIAs), because the attacking party is doing so merely to destroy the credibility of the other. A BRIA is *not* done to evade being held responsible for one's own mistake.

Implications of a `Sound` *Audit*

Theorem 3: Authentic high level logs of all jobs performed by a given guest G can be obtained by capturing low level logs at the interface between H and G, as well as ensuring a sound audit.

Proof: Theorem 2 states that high level logs can be obtained from low level logs.

Next, we prove the authenticity of the high level logs. For each job j performed by G,

(i) the integrity, non-repudiation, and atomicity properties are necessary and sufficient to certify the content of every message exchanged by H and G for a job j done by the latter, and hence to provide $IpCert_j$ and $OpCert_j$,

(ii) the timeliness, non-repudiation, and atomicity properties are necessary and sufficient to certify the times of transfer of every message exchanged by H and G for a job j done by the latter, and hence to provide $QoSCert_j$ and $QoECert_j$.

Thus, authentic high level logs are obtained.

Theorem 4: If low level logs are collected at every on-chip interface between mutually untrusting parties, and a sound low level audit is ensured, then accountability can be provided.

Proof:

- By Definition 2, responsibility is assigned at the granularity of organizations. As explained in Section IV-A, two guests cannot directly communicate with each other. The communication must pass through the host. Thus, all inter-organizational (in terms of design) boundaries on a chip are host-guest interfaces. The host and all the guest vendors do not trust each other (this is the worst case – complete mistrust). Thus, every host-guest interface involves two mutually untrusting parties.
- By Theorem 3, low level logs at an interface between H and a G, as well as a sound audit, ensures authentic high level logs of all jobs performed by G.
- Low level logging, and a corresponding sound audit, at every interface between mutually untrusting parties, implies authentic high level logs of all jobs performed by every guest.
- By Theorem 1, this is sufficient to provide accountability in the SoC.

Figure 3 pictorially depicts the overview of the solution, showing how low level logging at every host-guest interface, accompanied by a sound low level audit, gives authentic high level logs, and hence accountability.

We will now describe accountability solutions that vary in three aspects: (i) the class of SoC, (ii) the low-level auditing primitive applied, and (iii) the attack model.

B. Auditing in SoCs with Trusted Hosts

The first class of SoCs can be trivially audited as the host itself is a trusted entity. An example is illustrated in Figure 7(a). Note that the noc-vendor could be an untrusted guest or the host itself. In this class, no two guests (mutually untrusting parties) directly communicate with each other. All communication must pass through the trusted host. Thus, this is the "TTP as a mail server" primitive. The host can record the content of all messages to a guest ($IpLog = Ip$) and from a guest ($OpLog = Op$). It can record the times a job was issued to a guest and the time the result was

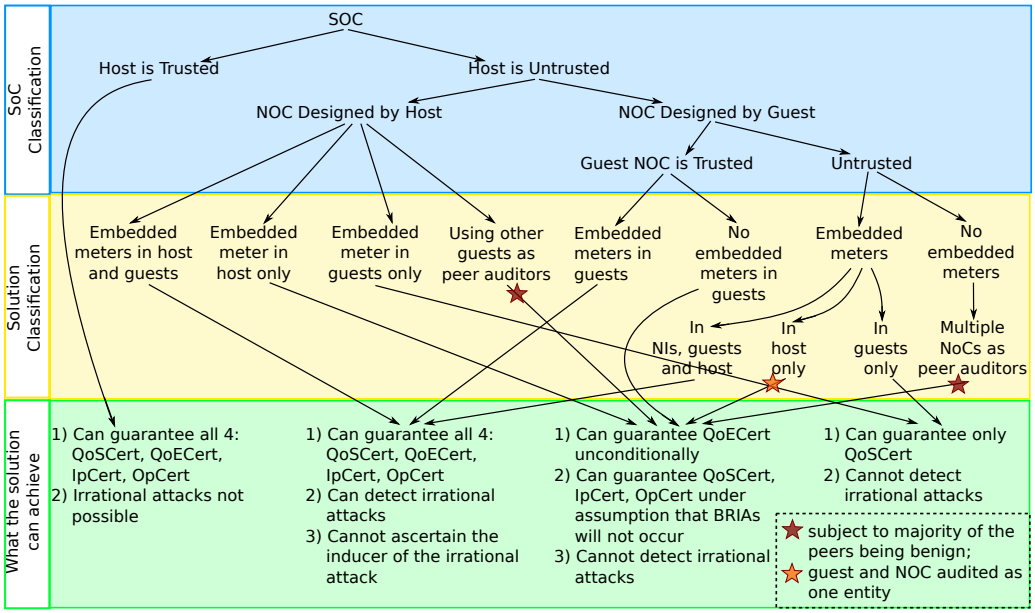


Fig. 6: Classification of 3PIP-containing SoCs and auditing solutions for each class

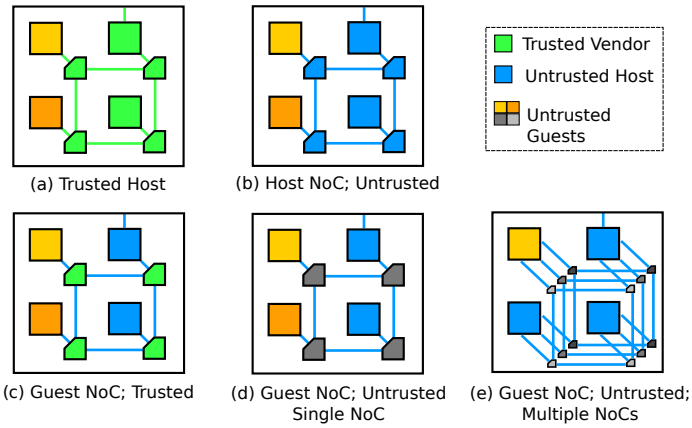


Fig. 7: Examples of Different SoC Classes

returned ($QoSLog = QoS$). For every resource request by the guest, the host can record the times of request and response ($QoELog = QoE$). Thus, accountability is provided, by Theorem 1.

C. Auditing in SoCs with the NoC Designed by an Untrusted Host

We deal with the class of SoCs with an untrusted host-designed NoC in this section. Figure 7(b) depicts an example. Three solutions are possible, with different constraints and benefits. We explain the solutions in great detail, as the intuitions gained here are useful in understanding the solutions for other classes of SoCs.

1) Redundant Metering: Embedded Meters at Both the Host and the Guests: The *DuoMeter* primitive is employed at every H-G interface, $\forall G \in \mathbb{G}$, as shown in Figure 8. Meters are embedded in both H and G. The *DuoMeter* primitive provides a `sound` audit, as discussed in Section VI-C. Thus, by Theorem 4, accountability is provided.

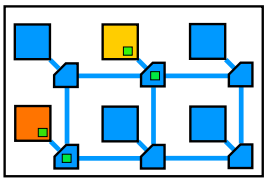


Fig. 8: Redundant Metering in SoCs

Game-Theoretic Reasoning about an Auditing Solution

Auditing solutions can be represented as *games*, and solving these games can help us reason about the solutions. The game-theoretic approach essentially helps us prove whether the application of an auditing primitive provides a `sound` audit. Though we have already proven that *DuoMeter* provides a `sound` audit, we employ the game-theoretic approach here to elucidate the latter's functioning. This discussion will aid the proving of more complex scenarios. We will limit our game-theoretic discussion to violations of the timeliness property, where delays are introduced. Extending the reasoning to other misbehaviors is straightforward.

The auditing solution certifies communication between two mutually untrusting parties – the host and the guest. These are the two players of the game. The two players may choose to either induce a delay (D), or not induce a delay (ND). These are the two strategies that the players may adopt. Thus, with two players, each with two strategy options, there are four possible outcomes. For each outcome, each player receives a certain pay-off (derives a certain advantage).

The pay-off captures the real world implications: (i) if it can be proven that a host has provided bad QoE, then it must be penalized. This is captured by a penalty of α. The guest is not penalized, as it has not committed any fault. (ii) Similarly, if it can be proven that a host has provided the expected QoE, but the guest has provided a poor QoS, then the guest is penalized by β. The host is not penalized. (iii) If any party – host or guest – invests in delay-inducing circuitry, this has an adverse

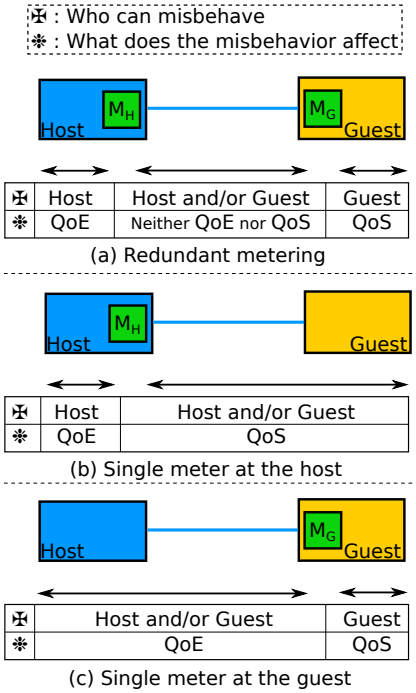


Fig. 9: Effect of misbehavior in the different schemes

effect on its design. It reduces the area budget, increases the design cost, and increases the power rating. This effect on the vendor is captured by a penalty of γ. Table I summarizes the rules that govern the payoffs for the host and the guest.

A player seeks to play that strategy that maximizes her own pay-off. An acceptable auditing solution is one where all players get the maximum pay-off by playing a strategy of *not delay* (ND). This would ensure that no undue delays affect the SoC, and every message exchange, both sending and receiving, is timestamped accurately. In other words, the timeliness property is ensured.

Each auditing solution in our formulation is represented by four games, which correspond to the ideal QoS and QoE being good or bad (good QoS refers to QoS satisfying $QoSCond$, bad QoS otherwise). If it can be proven that the dominating strategy, for both the players, in all four games, is to not induce a delay (ND), then the given auditing solution provides the timeliness property. If, similarly, the other properties can be proven, then a `sound` audit can be provided. As already proven in Theorem 4, a `sound` audit implies accountability.

We discuss the games for only a representative set of the proposed auditing solutions, owing to space constraints. It is possible to extend the reasoning to other solutions as well.

TABLE I: Payoff rules

Default: 0 for both Host and Guest
Bad QoE: $-\alpha$ for Host
Bad QoS, when the QoE is good: $-\beta$ for Guest
Penalty for spending on delay circuitry: $-\gamma$ for delaying entity
where α, β, γ are positive numbers

Figure 9(a) depicts the different communication paths (thick black arrows) that occur under the redundant metering scheme. The figure also depicts, for each communication path, who might induce a delay, and what the delay affects. In the path between H and M_H, only H may induce a delay. It will not do so however because such a delay worsens the QoE, and is therefore detrimental to H itself. Similarly, only G can induce a delay in the path between itself and M_G, and such a delay worsens the QoS. This is detrimental to G itself. The region between the two meters is interesting because any delay induced here can be detected by the meters, but the identity of the inducer cannot be ascertained. In such a scenario, since the chip's overall functionality suffers a delay, both the host and the guest suffer a penalty. Aside from these penalties, an additional penalty is levied if an entity has invested in delay-inducing circuitry – this models the strain on its area and power budgets. The payoff matrices for the host and the guest for all the four games are given in Table II.

Let us look at the first game: "good QoE - good QoS". Regardless of what G chooses to do, H gets a maximum pay-off by playing a strategy of ND. This is termed as a "strongly dominating strategy", in game-theory parlance. Similarly, G has a strongly dominating strategy of ND. Thus, as far as the first game is concerned, DuoMeter is an acceptable auditing solution. Likewise, the other three games also have a strongly dominating strategy of ND for both H and G. Thus, since H and G stand to suffer if they induce a delay, the timeliness property is provided by the DuoMeter strategy.

Similar games can be constructed to prove the other properties as well. Since a `sound` audit can be provided, accountability can be provided (Theorem 4).

2) Single Meter Embedded at the Host: Let us consider having a single embedded meter M_H at H. When H sends a job description to G, or responds to a resource request, the embedded meter is at the sender's side – the *MonoMeter-S* scheme. When G sends a job result to H, or makes a resource request, the embedded meter is at the receiver's side – the *MonoMeter-R* scheme. As explained in Section VI-C1, the integrity guarantee cannot be provided with a single embedded meter, since the certification is done only on one end. Thus, *IpCert* and *OpCert* cannot be unconditionally guaranteed. The time at which G started working on the job, and the time at which the job result was ready, cannot be audited fairly since the meter is embedded at H. Therefore, *QoSCert* cannot be unconditionally guaranteed. However, the time at which H started working on the resource request, and the time at which the resource response was ready, can be reliably audited. Therefore, *QoECert* can be guaranteed.

Though the single meter scheme seems deficient, we believe that it too is a useful design point in the space of auditing solutions. To understand this, we need to look closer at why the parties may misbehave (see Figure 9(b)). In the path between H and M_H, only H may misbehave. It will not do so because this would make its own computation wrong or delayed. In the path between M_H and G, both H and G may misbehave. Since the meter is at the host, it has the upper hand. If H or G modifies or drops a message, it is construed as a miscomputation on G's part. If H or G delays a message, the measured QoS is worsened. Thus, G has no incentive to misbehave.

TABLE II: Payoff matrix for the game "Redundant metering"

			Good QoS			Bad QoS		
Good QoE			Guest			Guest		
	Host		ND	D	Host	ND	D	
		ND	0,0	$0,-\gamma$		$0,-\beta$	$0,-\beta-\gamma$	
		D	$-\gamma$,0	$-\gamma,-\gamma$		$-\gamma,-\beta$	$-\gamma,-\beta-\gamma$	
Bad QoE			Guest			Guest		
	Host		ND	D	Host	ND	D	
		ND	$-\alpha$,0	$-\alpha,-\gamma$		$-\alpha$,0	$-\alpha,-\gamma$	
		D	$-\alpha-\gamma$,0	$-\alpha-\gamma,-\gamma$		$-\alpha-\gamma$,0	$-\alpha-\gamma,-\gamma$	

Now why would H modify, drop or delay messages in the channel between M_H and G? Firstly, it may choose to do so if this act allows it to deflect blame for its own wrong or slow functioning to G. By Axioms 1 and 3, G can be held responsible only if Op does not satisfy $OpCond_G$ and Ip satisfies $IpCond_G$, or QoS does not satisfy $QoSCond_G$ and QoE satisfies $QoECond_G$. For messages the host sends, the content and the time at which the message was ready, are reliably audited (*MonoMeter-S* scheme). Modifying or delaying sent messages after auditing does not improve $IpLog$ or $QoELog$. It merely worsens $OpLog$ or $QoSLog$. This is not beneficial to H. For messages H receives, modifying them implies message corruption as the contents of the message are encrypted. Here again, modifying or delaying packets does not improve $IpLog$ or $QoELog$, and so carries no benefit to H. The second scenario when H may choose to modify, drop or delay messages is when it is performing its functions correctly. The host has nothing to gain by performing such an attack. This is a *business related irrational attack* (BRIA), as defined in Section VII-A, where the host just wants to damage the guest vendor's credibility. We believe that such attacks are not very reasonable since apart from giving no immediate gain to H, the performance of the host network and the system performance as a whole are brought down.

Thus, under this assumption that BRIAs will not occur, having a single meter embedded at the host provides trustworthy *IpCert*, *OpCert* and *QoSCert* as well, in addition to *QoECert*. Thus, by Theorem 1, accountability is provided.

Game-Theoretic Reasoning

The above arguments can be formally analyzed through a game-theoretic formulation. Figure 9(b) shows two regions of operation when the meter is embedded in H. The first region is where H is working. Here, only H may induce a delay, but it will not do it because this makes the QoE worse. The second region is more interesting and will form the focus of the game. Here, the guest is working and the two parties are communicating. Here, both H and G may induce delays. A delay makes the *measured* QoS worse (since the meter is on the host's side, the communication gets counted towards QoS as well). The payoff rules are again as given in Table I.

The game matrices for this scenario are given in Table III. In all the four games, the dominating strategy for both the host and the guest is *not delay* (ND). Either party inducing a delay adversely effects both parties (with the guest suffering more in many cases). Thus, single metering at the host provides the timeliness property. Similar games can be constructed to reason about the other properties as well. Thus, a `sound` audit can be provided, and consequently accountability can be provided, by Theorem 4.

Thus, the seemingly non-intuitive solution of having a *single* meter at each interface, provides accountability. Though the solution, at first sight, appears to unfairly favor the host, the formal analysis conclusively proves that this is indeed a valid accountability solution. The implications of this are large. The amount of chip area spent on meters directly comes down by half. We can expect a similar reduction in the power consumption as well. This can be a great enabler in adopting the solution.

We can also see how BRIA is captured in this framework. In the "good QoE - good QoS" game, H may induce a delay and make it appear that G provided bad QoS. G thus incurs a penalty of β. However, H itself incurs a penalty of γ for investing in delay circuitry. Thus, rational behavior of H, which requires that it maximizes its own pay-off, leads to it not attempting such an attack.

The strength of the two auditing solutions: redundant metering and single meter at the host, must also be noted. The relative values of α, β, γ, and δ do not matter. As long as the four are positive values, both these solutions provide a `sound` audit.

3) Single Meter Embedded at the Guest: Having a single meter M_G embedded at G is another option. When H sends a job to G, the latter can begin working on the job only after it gets the message audited at M_G. Thus, the job description and the time at which G starts working on the job are reliably audited. Once the guest has finished the job, it gets the results audited at M_G. Thus, the results and the time of completion of the job are reliably audited. Since the start and end times of the job execution are reliably audited, *QoSCert* can be reliably provided.

The same cannot be said for the other three certificates. Consider the case when QoS does not satisfy $QoSCond_G$ and QoE satisfies $QoECond_G$. This scenario warrants G being held responsible (Axiom 3). However, since the meter is embedded at G, it may induce delays that result in the $QoELog$ not satisfying $QoECond_G$ (see Figure 9(c)). This allows the guest to escape punitive action for underperforming. Since a party (G) stands to directly benefit from misbehaving, this auditing solution *cannot* provide a `sound` audit (Definition 6), and consequently accountability (Theorem 4). The same line of reasoning can be applied for G modifying or dropping messages as well.

TABLE III: Payoff matrix for the game "single meter at the host"

			Good QoS				Bad QoS	
			Guest				Guest	
Good QoE	Host		ND	D	Host		ND	D
		ND	0,0	$0,-\beta-\gamma$		ND	$0,-\beta$	$0,-\beta-\gamma$
		D	$-\gamma,-\beta$	$-\gamma,-\beta-\gamma$		D	$-\gamma,-\beta$	$-\gamma,-\beta-\gamma$
Bad QoE			Guest				Guest	
	Host		ND	D	Host		ND	D
		ND	$-\alpha$,0	$-\alpha,-\gamma$		ND	$-\alpha$,0	$-\alpha,-\gamma$
		D	$-\alpha-\gamma$,0	$-\alpha-\gamma,-\gamma$		D	$-\alpha-\gamma$,0	$-\alpha-\gamma,-\gamma$

TABLE IV: Payoff matrix for the game "Single meter at the guest"

			Good QoS				Bad QoS	
			Guest				Guest	
Good QoE	Host		ND	D	Host		ND	D
		ND	0,0	$-\alpha,-\gamma$		ND	$0,-\beta$	$-\alpha,-\gamma$
		D	$-\alpha-\gamma$,0	$-\alpha-\gamma,-\gamma$		D	$-\alpha-\gamma$,0	$-\alpha-\gamma,-\gamma$
Bad QoE			Guest				Guest	
	Host		ND	D	Host		ND	D
		ND	$-\alpha$,0	$-\alpha,-\gamma$		ND	$-\alpha$,0	$-\alpha,-\gamma$
		D	$-\alpha-\gamma$,0	$-\alpha-\gamma,-\gamma$		D	$-\alpha-\gamma$,0	$-\alpha-\gamma,-\gamma$

Game-Theoretic Reasoning

The game-theoretic formulation helps us to formally reason about this scheme. Figure 9(c) shows two regions of operation. The first is where H is working and the two entities are communicating. Here, both H and G may induce delays. The delay makes the *measured* QoE worse (since the meter is on the guest side, the communication gets counted towards QoE as well). The game will focus on this region. The second region is where G is working. Here, only the guest may induce a delay, but it will not do so because it makes the QoS worse. The payoff rules are summarized in Table I.

The game matrices for this scenario are given in Table IV. The dominant strategy for H, in all the four games, is *not delay* (ND). If the host plays ND, then the guest maximizes its payoff by playing *not delay* (ND) in games 1, 3 and 4 – "good QoE - good QoS", "bad QoE - good QoS", "bad QoE - bad QoS". In game 2 – "good QoE - bad QoS" – however, the guest's choice of strategy depends on the relative values of the penalties for poor QoE (β) and spending on delay circuitry (γ). If $\beta > \gamma$, that is, if the gain through avoiding the penalty for bad QoS outweighs the loss of investing in delay circuitry, then the guest maximizes its payoff by playing *delay* (D). Since an entity (G) benefits from maliciously delaying under a particular scenario, single metering at the guest *cannot* provide the timeliness property, and consequently, a `sound` audit (Definition 6).

4) No Embedded Meters: The *ConstrainedPeer* primitive can be used in this scenario. The constrained solution needs to be employed as the host controls all the links. The intermediaries that provide certification are the other guests in the SoC. If the assumption can be made that a maximum of k guests can be malicious, then all communication between the host and the guest is routed through at least $2k+1$ other guests. These $2k+1$ guests provide the certificates. It was reasoned in Section VI-D2 that *ConstrainedPeer* achieves what *MonoMeter* does, under the k-maximum adversary assumption. Only the host has direct access to the certifying entities, that is, the other guests. Thus, this can be analyzed as a single embedded meter scheme, with the meter embedded at the host

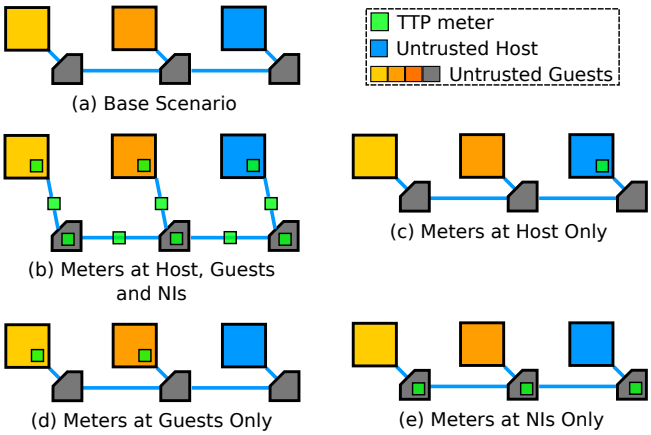


Fig. 10: Auditing solutions for SoCs with untrusted guest-NoC

(see Section VII-C2). Consequently, accountability is provided under the assumption that BRIAs will not occur.

D. Auditing in SoCs with a Guest-NoC Trusted by the Host and All Other Guests

An example is illustrated in Figure 7(c). Two solutions are possible.

1) Embedded Meters in the Guests: The *DuoMeter* primitive can be achieved at every interface between two mutually distrusting parties by embedding TTP meters in the guest components. *DuoMeter* provides a `sound audit`, and so accountability can be provided.

2) No Embedded Meters in the Guests: This scenario is similar to having a single meter embedded in the host, that is the *MonoMeter* primitive, as discussed in Section VII-C2. Thus, accountability can be provided, under the assumption that BRIAs will not occur.

E. Auditing in SoCs with a Guest-NoC Untrusted by the Host and All Other Guests

1) Embedded Meters in the Host, Guests and Network Interfaces: Figure 10(a) shows an example of an SoC with

an untrusted guest-NoC. Any communication between the mutually untrusting host and guest involves a third untrusted party – the NoC. A multitude of auditing schemes are possible – a few important designs will be discussed here.

One solution is to employ the *DuoMeter* primitive at every interface between untrusting parties. Figure 10(b) depicts the example SoC under this scheme. Accountability, of all components including the guest NoC, is provided.

Another solution is to embed meters in host cores only, as shown in Figure 10(c), with a symmetric key cryptosystem set up between the embedded meter and the guest. This resembles the scenario described in Section VII-C2, with the NoC's working also counting towards the guest's audit. Both the guest and the NoC are penalized for any miscomputation, underperformance or malice on the part of either the guest or the NoC. Thus, under the assumption that BRIAs will not occur, accountability is provided, with the NoC and the guest being audited as a single entity.

Having meters embedded only in the guests, as shown in Figure 10(d), can provide only a reliable *QoSCert*. The situation is similar to that described in Section VII-C3. Embedding meters only in the network interfaces, as shown in Figure 10(e), does not achieve any facet of auditing. Neither the host's working nor the guest's working can be reliably audited. The NoC's working cannot be audited either.

2) No Embedded Meters: Without embedded meters, no auditing of any degree is possible unless multiple NoCs are available, each provided by a different vendor (see Figure 7(e)). The *ConstrainedPeer* primitive may then be applied. The assumption now is that at most k malicious NoCs are present. Now messages are sent over $2k+1$ NoCs such that the minimum of $k+1$ benign NoCs provide the certificates. Since the host has direct access to the NoCs, this scheme is similar to the scenario discussed in Section VII-D2, under the k-maximum adversary assumption. Accountability is provided under the assumption that BRIAs will not occur.

Figure 6 summarizes the auditing solution space. For each type of SoC, multiple solutions are possible, that achieve auditing to different degrees, and entail different costs.

VIII. Practical Considerations

The focus of this paper is to propose a formal approach to construct an accountability solution and to reason about its viability. However, from a practical standpoint, it is also important that the accountability solution be efficiently implemented – in terms of performance, power, and area. Such a practical implementation has been discussed by Kalayappan et al. [14], whose solution falls in the *DuoMeter* category, that was discussed in Section VII-C1. The paper [14] also demonstrates the feasibility of their approach using example bugs from the published errata documents of some popular commercial SoCs.

Auditing

Lightweight cryptographic modules are used in the design of the TTP meters. This ensures that the overhead of the auditing infrastructure is minimal: 0.49% in terms of performance, 0.194% in terms of area, and 0.99μW in terms of power.

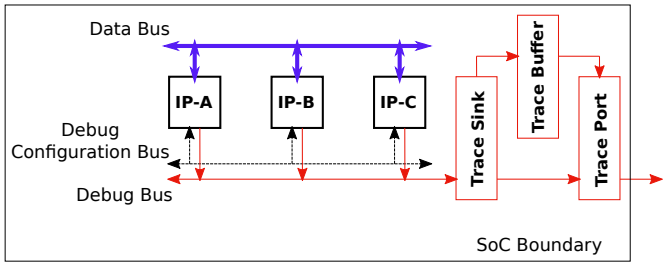


Fig. 11: Design for Debug Architecture in SoCs

Logging

Overview of Design-for-Debug Architectures: For logging, already existent design-for-debug (DfD) structures [36][37] may be re-purposed. Figure 11 illustrates the DfD architecture of an SoC. The DfD architecture primarily serves to increase visibility into the inner workings of the chip during the stage of post-silicon validation, that is, the validation of the chip before it is commissioned. When the chip under test does not behave as per the specifications, the test engineers are expected to localize the source of the error. This is extremely hard to do given only the external input and output pins of a chip – this is known as the "limited observability problem" [38]. The DfD hardware serves to give the test engineer access to the interior signals of the chip, thus enabling the tracing of the source of the bug. This is analogous to inserting print statements within a software program while debugging it. The debug configuration bus is used to configure which inner signals in the chip are to be captured, and under what conditions. The captured signals, called the *debug trace*, is sent off-chip via the trace port. The test engineers study this trace to root-cause the chip's erroneous behavior.

In certain scenarios, the volume of the trace generated may be too large resulting in slowing down of the chip. This slows down the validation activity. To overcome this, the DfD architecture typically provides various techniques [38] to reduce the trace volume such as spatial summarization [39], temporal summarization, and fingerprinting, to name a few. The choice of technique is context-dependent – the nature of the module under observation and that of the bug being studied determine the choice. Vermeulen et al. [26] have demonstrated that such additional logic results in a mere 0.2% overhead in terms of area.

Re-purposing the DfD Architecture for Accountability: After the chip is commissioned, the DfD hardware is traditionally unused. We propose to re-purpose this hardware to collect the logs required for accountability. For every message transferred between the host and the guest, a duly summarized (using one of the aforementioned summarization techniques) message is saved via the trace port, along with the certificate provided by the TTP meter. In case of the top-level host, that is, the System Integrator, the logs are stored in an off-chip disk. At all other levels, the logs are handed over to the host at the immediate

higher level. In case of a dispute in the field, the trusted third party arbitrates it using the stored certified logs. The logs are analyzed to localize the root cause of the chip's erroneous behavior to one particular on-chip module. The corresponding vendor is then held accountable. Note that the cryptographic measures set in place prevent the tampering of logs, as well as the spurious addition or deletion logs.

Re-purposing of DfD hardware for other purposes after commissioning is a popular area of research [40]. Particularly, researchers are actively proposing the use of vestigial DfD hardware to implement security policies in the face of untrusted third party IPs [41].

IX. Conclusion

This work presents a thorough formal treatment of the notion of accountability and the space of accountability solutions. Based on the principle that authentic low level logs can achieve accountability [14], we first proposed an array of low level auditing primitives. We then proposed a classification of modern SoCs that demand different approaches to accountability, and may be susceptible to different attack models. We presented an exhaustive comparison of the effects of applying the different primitives to the different SoC models, under the threat of different attack models. We also proposed a formal game-theoretic methodology to reason about an accountability solution. A formal proof of a solution is essential to convince the concerned parties to adopt it. The formal approach helped us to prove the viability of the most generic solution that makes no assumptions regarding the SoC or the attack model. The formal approach also helped prove the viability of another solution, one that is quite non-intuitive (single meter at host), and highly efficient.

References

[1] "The athena group," http://www.athena-group.com/.

[2] "Invia," https://www.invia.fr/.

[3] "Synopsys symmetric cryptographic engines," https://www.synopsys.com/designware-ip/security-ip/cryptography-ip/symmetric-cryptographic-engines.html.

[4] "Intel custom foundry," https://www.intel.com/content/www/us/en/foundry/overview.html.

[5] J. Villasenor and M. Tehranipoor, "Chop shop electronics," *IEEE Spectrum*, vol. 50, no. 10, pp. 41–45, 2013.

[6] A. Vörg, M. Radetzki, and W. Rosenstiel, "Measurement of ip qualification costs and benefits," in *DATE*, 2004.

[7] L. Wang and H. Luo, "Automated ip quality qualification for efficient system-on-chip design," in *ICEPT-HDP*, 2012.

[8] D. D. Gajski, A. C.-H. Wu, V. Chaiyakul, S. Mori, T. Nukiyama, and P. Bricaud, "Embedded tutorial: essential issues for ip reuse," in *ASPDAC*, 2000.

[9] D. Karlsson, P. Eles, and Z. Peng, "Formal verification of component-based designs," *Design Automation for Embedded Systems*, 2007.

[10] H. Bouaziz, S. Chouali, A. Hammad, and H. Mountassir, "Sysml blocks adaptation," in *Formal Methods and Software Engineering*, 2015.

[11] S. Adee, "The hunt for the kill switch," *iEEE SpEctrum*, 2008.

[12] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware trojans in third-party digital ip cores," in *HOST*, 2011.

[13] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," *ACM TODAES*, 2016.

[14] R. Kalayappan and S. R. Sarangi, "Providing accountability in heterogeneous systems-on-chip," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 5, p. 83, 2018.

[15] "Trusted computing group." [Online]. Available: https://trustedcomputinggroup.org/

[16] "Siidtech rolls out silicon fingerprinting for chip tracing, fraud fighting," https://www.eetimes.com/siidtech-rolls-out-silicon-fingerprinting-for-chip-tracing-fraud-fighting/#.

[17] A. Kupcu, "Incentivized outsourced computation resistant to malicious contractors," *IEEE TDSC*, 2017.

[18] J. Graf, "Trust games: How game theory can guide the development of hardware trojan detection methods," in *HOST*, 2016.

[19] S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, and Q. Wu, "A survey of game theory as applied to network security," in *HICSS*, 2010.

[20] R. Kalayappan and S. R. Sarangi, "A survey of checker architectures," *ACM CSUR*, 2013.

[21] L. Spainhower and T. A. Gregg, "Ibm s/390 parallel enterprise server g5 fault tolerance: A historical perspective," *IBM Journal of Research and Development*, 1999.

[22] T. M. Austin, "Diva: A reliable substrate for deep submicron microarchitecture design," in *MICRO*, 1999.

[23] C. Liu, J. Rajendran, C. Yang, and R. Karri, "Shielding heterogeneous mpsocs from untrustworthy 3pips through security-driven task scheduling," *IEEE TETC*, 2014.

[24] J. Rajendran, H. Zhang, O. Sinanoglu, and R. Karri, "High-level synthesis for security and trust," in *IOLTS*, 2013.

[25] M. Abramovici and P. Bradley, "Integrated circuit security: New threats and solutions," in *CSIIRW*, 2009.

[26] K. Goossens, B. Vermeulen, R. Van Steeden, and M. Bennebroek, "Transaction-based communication-centric debug," in *NOCS*, 2007.

[27] B. Vermeulen, "Functional debug techniques for embedded systems," *IEEE Design & Test of Computers*, 2008.

[28] R. Kalayappan and S. R. Sarangi, "Secx: A framework for collecting runtime statistics for socs with multiple accelerators," in *ISVLSI*, 2015.

[29] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *2007 44th ACM/IEEE Design Automation Conference*. IEEE, 2007, pp. 9–14.

[30] S. Amir, B. Shakya, D. Forte, M. Tehranipoor, and S. Bhunia, "Comparative analysis of hardware obfuscation for ip protection," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM, 2017, pp. 363–368.

[31] Y. Liu, C. Bao, Y. Xie, and A. Srivastava, "Introducing tfue: The trusted foundry and untrusted employee model in ic supply chain security," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[32] K. Vaidyanathan, B. P. Das, E. Sumbul, R. Liu, and L. Pileggi, "Building trusted ics using split fabrication," in *2014 IEEE international symposium on hardware-oriented security and trust (HOST)*. IEEE, 2014, pp. 1–6.

[33] S. Kremer, O. Markowitch, and J. Zhou, "An intensive survey of fair non-repudiation protocols," *Computer communications*, 2002.

[34] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *CHES*, 2007.

[35] K. Argyraki, P. Maniatis, O. Irzak, S. Ashish, and S. Shenker, "Loss and delay accountability for the internet," in *ICNP*, 2007.

[36] P. Jayaraman and R. Parthasarathi, "A survey on post-silicon functional validation for multicore architectures," *ACM CSUR*, 2017.

[37] N. Stollon, *On-chip instrumentation: design and debug for systems on chip*. Springer Science & Business Media, 2010.

[38] F. F. Prabhat Mishra, *Post-Silicon Validation and Debug*, 1st ed. Springer International Publishing, 7 2019.

[39] S. Chandran, P. R. Panda, S. R. Sarangi, A. Bhattacharyya, D. Chauhan, and S. Kumar, "Managing trace summaries to minimize stalls during postsilicon validation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 6, pp. 1881–1894, 2017.

[40] N. Jindal, S. Chandran, P. R. Panda, S. Prasad, A. Mitra, K. Singhal, S. Gupta, and S. Tuli, "Dhoom: Reusing design-for-debug hardware for online monitoring," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: ACM, 2019, pp. 99:1–99:6. [Online]. Available: http://doi.acm.org/10.1145/3316781.3317799

[41] A. Basak, S. Bhunia, and S. Ray, "Exploiting design-for-debug for flexible soc security architecture," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 167.

Rajshekar Kalayappan is an Assistant Professor in the Department of Computer Science and Engineering, IIT Dharwad, India. He graduated with a Masters and Ph.D degree in Computer Science from IIT Delhi in 2012 and 2017 respectively. He has a Bachelors degree in Information Science from Visveswaraya Technological University, Belgaum. His research interests include computer architecture, fault-tolerant systems, and hardware security.

Smruti R. Sarangi is an Associate Professor in the Department of Computer Science and Engineering, IIT Delhi, India. He has spent four years in industry working in IBM India Research Labs, and Synopsys. He graduated with a M.S and Ph.D in computer architecture from the University of Illinois at Urbana-Champaign in 2007, and a B.Tech in computer science from IIT Kharagpur, India, in 2002. He works in the areas of computer architecture, parallel and distributed systems. Prof. Sarangi is a member of the IEEE and ACM.