Position Paper: A Case for Machine-Checked Verification of Circumvention Systems

Vitor Pereira
Algoritmo Sapiente, LDA
Portugal
vitorm2p@gmail.com

Ahmed Irfan
SRI
USA
ahmed.irfan@sri.com

Vinod Yegneswaran
SRI
USA
vinod@csl.sri.com

Nick Feamster
University of Chicago
USA
feamster@uchicago.edu

Prateek Mittal
Princeton University
USA
pmittal@princeton.edu

Vitaly Shmatikov
Cornell Tech
USA
shmat@cs.cornell.edu

Abstract

We present a case for adopting formal verification as a methodology for evaluating hidden communication systems (HCS), with the goal of supplementing the current ad hoc and informal analysis strategies used in state-of-the-art HCS with a rigorous pathway to the analysis of these systems. Our position is that the core security properties of these systems, like resistance to traffic analysis and detectability, are fundamentally *distinguishability* problems aligned with the same distinguishability concepts found in cryptography. Thus, we propose the lifting of cryptographic results to the HCS domain and the leverage of the existing formal verification infrastructure available for cryptography to derive concrete security and performance bounds for HCSs. Concretely, we propose the usage of simulation-based cryptography as a rigorous framework to model interactions between users, censors, and network protocols, allowing precise definitions of censorship relevant goals such as unobservability, unblockability, covertness, and performance guarantees. We showcase the utility of this formal approach by presenting VeriWeird, an envisioned tool that bridges the formal gap existing in state-of-the-art HCS and demonstrate our approach through a (preliminary) analysis of meek. Our case study highlights the potential of machine-checked verification to provide strong, systematic assurances of resilience and also identify vulnerabilities in adversarial environments.

Keywords

formal verification, simulation-based cryptography, hidden communication systems, censorship circumvention

1 Introduction

HCSs exploit underspecified protocols by taking advantage of ambiguities or undefined ("weird") behavior in specifications to bypass censorship [10]. These systems leverage the fact that some network protocols may not fully define how certain data or signals should be handled, allowing them to manipulate traffic in ways that evade detection by censors. Example strategies include domain fronting, packet fragmentation and header manipulations [5], traffic substitution [3, 13, 14], and steganographic embedding in unexpected cover protocols. Notably, the process of developing and fielding censorship circumvention tools has been largely ad-hoc and devoid of scientific rigor [8], leading to a cat-and-mouse game.

For example, several recently proposed covert channels substitute or replace traffic generated by a cover application and send it through the application's encrypted network connection. Their security arguments involve pre-trained network-trace classifiers not distinguishing covert from those of the cover application. Unfortunately, many of these approaches fail against application-aware adversaries who observe or induce discrepancies such as violations of application-specific invariants.

In this position paper, we contend that the core security goals of circumvention systems — undetectability, mimicry, resistance to fingerprinting — as well as their performance properties — latency, throughput and goodput — are fundamentally distinguishability properties, analogous to those long studied in cryptographic contexts, and propose VeriWeird (Figure 1), as a framework for integrating foundational tools and techniques for rigorous modeling and analysis of HCSs used for censorship circumvention.

VeriWeird's key innovation is the concept of *simulatable indistinguishability proofs* (SIPs), a novel approach to the formalization of HCSs that is based on the encoding of HCS properties in the simulation-based cryptographic proof paradigm [11]. The goal of simulation-based cryptography is to establish probabilistic indistinguishability between an *ideal world*, where the primitive is *secure* by definition, and the *real world*, where concrete protocols are executed. In terms of HCS, the real world represents the execution of the HCS channel (like Balboa) and the ideal world represents the application channel (like WebRTC) on top of which the HCS channel of the real world is defined. The objective is to show the existence of a *simulator* that the ideal world can use to generate a plausible execution trace of the application channel that matches the real-world traces from the HCS channel. If the traces produced by the real world are indistinguishable from those produced by the ideal world, the protocol is *secure* since security is built into the definition of the ideal world. Because the blocking of HCSs is based on detecting discrepancies between traces generated by its execution and the ideal-world application, our simulation-based approach provides an apposite proof paradigm.

Proceedings on Privacy Enhancing Technologies YYYY(X), 1–5
© YYYY Copyright held by the owner/author(s).
https://doi.org/XXXXXXX.XXXXXXX

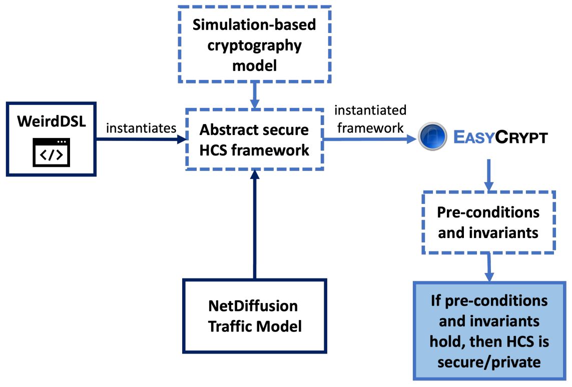
Simulation-based cryptography model
WeirdDSL
instantiates
Abstract secure HCS framework
instantiated framework
EasyCrypt
Pre-conditions and invariants
If pre-conditions and invariants hold, then HCS is secure/private
NetDiffusion Traffic Model

Figure 1: VeriWeird workflow. Blue arrows and boxes represent the workflow of security and privacy proofs.

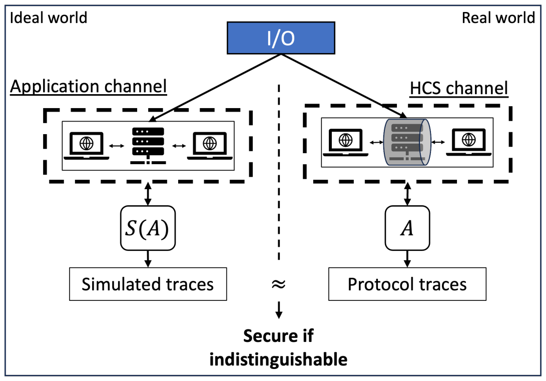

Figure 2: VeriWeird's simulation-based cryptography approach

The conceptual usage of simulation-based cryptography to model HCS security suggests that the same computer-aided cryptography [2] proof infrastructure that exists for cryptographic proofs can also be applied to the analysis of HCS, providing rigorous methodologies to mathematically prove and establish concrete results about HCS. However, we have yet to see meaningful adoption of such techniques in the censorship circumvention literature [?].

Notably, applying formal verification to censorship circumvention introduces a myriad of new and interesting research challenges. In contrast to cryptographic proofs, which rely on well-defined ideal functionalities, circumvention proofs require accurate and context-specific models of network traffic—both benign traffic of cover applications and generic background traffic. For example, applications that use WebRTC exhibit highly divergent traffic patterns despite using the same underlying protocol, and video streams may tolerate significant frame loss without perceptual degradation—factors that must be captured in the modeling stage to support meaningful proofs. We argue that indistinguishability claims in this space *must be proved*, not just empirically demonstrated through classifier performance on custom traces of network traffic.

2 Modeling HCS via simulation-based cryptography

Adversary modeling is a key gap in the field of HCSs: Security arguments for existing circumvention systems are based on ad-hoc adversary models, often incomplete or not representative of real-world adversaries, resulting in allegedly secure designs that fail even against relatively straightforward attacks. Adversarial models for detecting circumvention represent different capabilities and approaches that detectors might use to monitor and control network traffic. These adversaries range from simple, stateless techniques to more complex, cross-layer and cross-flow techniques that allow for deeper inspection and comprehensive examination of circumvention mechanisms.

The workflow of an HCS simulation-based proof (Figure 2) is as follows. The adversary is responsible for initializing the protocol, generating initial data and providing inputs for each participating dataset. It also chooses if it wants to *corrupt* any party, gaining control over its execution. The adversary will then interact with either the real world or the ideal world. The real world is responsible for executing the HCS channel, producing a set of HCS channel network traces T_{HCS} together with the output O_{HCS} of each party participating in the HCS communication. Second, the ideal world will execute the application channel as if no censorship mechanism is in place (messages can be sent without covertness) which will produce an output $O_{application}$ for each party. The ideal world will then invoke a *simulator* that will attempt to reproduce the online behavior of the HCS channel and produce protocol traces $T_{simulator}$. The simulation proof is completed by trying to distinguish between O_{HCS} and $O_{application}$ and between T_{HCS} and $T_{simulator}$:

- if $O_{HCS} \sim O_{application}$, then the HCS channel correctly transmitted the message it was intend to transmit; and
- if $T_{HCS} \sim T_{simulator}$, then the traces produced by the HCS channel do not expose the HCS communication and the HCS message was securely transmitted because the HCS communication was not detected.

Because we are encoding HCS properties as cryptographic results, the aforementioned equivalences will be established via probabilistic reasoning. One advantage of adopting the simulation-based cryptography proof paradigm is that it is able to model all properties that can be defined by comparing traces, which make it an attractive methodology to assess circumvention protocols since blocking of covert channels is based on detecting discrepancies between the application channel traces and the HCS channel traces. A second advantage of formalizing HCS properties via simulation is that the paradigm is parametric in the adversaries and can, therefore, tolerate any adversary (passive or active). In fact, in simulation-based cryptography, the real and ideal world are fixed and it is the adversary that changes; i.e., it is the power given to the adversary that defines the security level of the protocol.

3 VeriWeird Approach to Modeling HCSs

VeriWeird rigorous approach to the analysis of HCSs incorporates a domain-specific language (DSL), dubbed WeirdDSL– an extension of the existing P[1] language [6] for modeling of network and distributed systems (i.e., event-driven systems), and a formal verification backend that integrates the P [12] analysis engine and the EasyCrypt [4] cryptographic proof assistant.

The envisioned workflow of VeriWeird is depicted in Figure 1. VeriWeird integrates HCS concepts embedded in the P language

[1]P language is extensively used by AWS, Microsoft and Academia.

and the Kinetic language with a robust and scalable proof pipeline based on EasyCrypt. First, a WeirdDSL specification of an HCS is *compiled* into what we denote by *abstract secure HCS framework*, i.e., an abstract definition of what an HCS is and of its security definitions written in a simulation-based cryptographic fashion. This framework is formalized in EasyCrypt and is agnostic of the HCS being analyzed, meaning that the framework is able to accommodate multiple instances of HCSs. The instantiated EasyCrypt framework produces a set of pre-conditions and invariants that need to be met in order for an HCS to achieve the desired level of security and privacy according to some well-defined adversary. In addition to WeirdDSL definition of an HCS, VeriWeird will produce the *security and privacy* proofs described at the bottom of Figure 1. Essentially, we prove that, if pre-conditions and invariants hold, then the HCS achieves the desired level of security and privacy, according to some adversary.

NetDiffusion [9] is a powerful probabilistic modeling tool designed to generate realistic synthetic network traffic based on real-world datasets. In scenarios where we want to evaluate the detectability of circumvention tools like Meek, it becomes crucial to model background traffic accurately, since an adversary's false positive rate depends heavily on the characteristics of benign traffic. By training a NetDiffusion model on packet-level traces—such as those collected from campus networks, backbone providers, or public datasets—we can learn the statistical distributions of traffic features like packet sizes, interarrival times, domain names etc. Once trained, the model can sample large volumes of synthetic traffic that mirrors the complexity and diversity of real traffic. These samples can then be analyzed to extract empirical distributions—for example, estimating the mean and variance of packet sizes—which are crucial for simulating the background traffic in a formal model.

4 Modeling meek Detection

meek [7] is a particular example of a general class of HCSs that arise from underspecified protocols and protocol ambiguities. meek uses domain fronting to hide the target bridge relay behind popular cloud service domains, like google.com or amazon.com, by using different domains in the unencrypted and encrypted parts of an HTTPS request. Specifically, the server name indicator (SNI) seen by the censors is a permitted site, while the true destination is hidden inside the encrypted request. This allows for Tor traffic to look like normal HTTPS traffic destined to the permitted site and has the side effect of blocking meek be very expensive for censors.

In this model, we consider an adversary that attempts to detect and block meek traffic using two signals. The **SNI value** if it matches the known meek front domain; and the **traffic profile**: specifically, the packet size distribution associated with the traffic. We assume that: (1) meek traffic follows a **normal distribution** with mean packet size of 512 bytes and variance 30. (2) Background traffic packet size follows a **Poisson distribution** with $\lambda = 1024$. (3) Only 0.01% of background traffic uses the front domain (to simulate overlap with benign uses).

The adversary's detection rule is conjunctive: classify traffic as meek only if both the SNI and traffic pattern match known meek signatures.

```
spec Confidentiality observes eTLSPacketRequest,
      eHTTPSPacketRequest {
  start state Init { /* do the initialization */ }
  state Distinguish {
    /* check assertion about payloads associated with observed
         events */ }
}
spec Anonymity observes eTLSPacketRequest,eHTTPSPacketRequest {
  start state Init { /* do the initialization */ }
  state Detect {
    /* check assertion about payloads associated with observed
         events */ }
}
```

Figure 3: WeirdDSL definition of channel confidentiality and anonymity properties against the SNI monitoring adversary.

```
machine CensoredClient {
  ..
  start state Init { /* do the initialization */ }
  state StartCommunication {
    on eHandshakeRequest do {
      /* send TLS packet to network with SNI information */ }
  }
  state WaitingHandshake {
    on eHandshakeResponse do {
      /* update handshake variable and go to Sending */ }
  }
  state Sending {
    on eEncryptedRequest do {
      /* send encrypted data to network, go to Wait4Response */ }
  }
  state Wait4Response {
    on eEncryptedResponse do {
      /* process data and go to Sending */ }
  }
}
machine Adversary {
  start state Init { }
  state SNIMonitoring {
    /* monitor TLS-SNI field to detect meek traffic */ }
   state TrafficProfiling {
    /* monitor packet sizes to detect meek traffic */ }
}
```

Figure 4: High-level definition of a censored client and an adversary in WeirdDSL. Machines are data structures where it is possible to define events that occur given some action.

4.1 Modeling in WeirdDSL

The properties of meek that we want to prove are stated in Figure 3. The first property is channel confidentiality, where the adversary will try to *distinguish* between traffic generated either by the application channel or by the HCS channel. The second property is anonymity, where the adversary will try to *detect* the identity of some user based on the traffic it sees. Note that these properties are defined against the SNI monitoring adversary specified in Figure 4. Client, network and adversaries are defined as P *machines* in the WeirdDSL. For example, Figure 4 shows how a censored client and the adversary can be modeled in VeriWeird.

We assume a tool like NetDiffusion is used to simulate background traffic. Once packet sizes are sampled from NetDiffusion, one might compute parameters like mean and standard deviation and use these parameters to define an approximation in EasyCrypt—for instance, using a normal distribution centered around the observed mean ± one standard deviation. This refined distribution replaces toy assumptions in the background traffic model, making the detection analysis more realistic. This approach provides a principled pipeline: train NetDiffusion on background traces, sample synthetic traffic, extract traffic profile distributions, integrate them into formal models, and verify detection bounds. The result is a hybrid methodology that blends statistical realism with provable guarantees—enabling more credible evaluations of censorship resistance strategies.

4.2 Modeling in EasyCrypt

```
op get_sni : domain -> sni.
module Distr = {
  proc Meek_profile_dist() : traffic_profile = {
    base <$$ uniform(497, 527); // Normal approx
    return base;
  }.
  proc background_profile_dist() : traffic_profile = {
    base <$$ uniform(1000, 1050); // Poisson-like
    return base;
  }.
  proc background_domain_dist() : domain = {
    r <$$ bernoulli(0.0001);
    return if r then front_domain else background_domain;
  }. }.
module type TLS_ENCRYPTION = {
  proc encrypt(d: domain, m: message) : ciphertext
}.
module Adversary(Enc : TLS_ENCRYPTION) = {
  proc is_meek_pattern(profile : traffic_profile) : bool = {
    return (profile >= 497) && (profile <= 527);
  }
  proc detect_meek(s : sni, p : traffic_profile) : bool = {
    if s = get_sni(front_domain) then {
      return is_meek_pattern(p);
    } else { return false; }
  } }.
module MeekEnc : TLS_ENCRYPTION = {
  proc encrypt(d: domain, m: message) : ciphertext = {
    c.sni_info <- get_sni(d);
    if m = real_msg then {
      c.profile <@ Distr.meek_profile_dist();
    } else {
      c.profile <- Distr.background_profile_dist();
    };
    c.payload <- m;
    return c;
  } }.
module Game1(Enc : TLS_ENCRYPTION) = {
  proc main() : bool = {
    d <- front_domain; (* domain *)
    m <- real_msg; (* message *)
    c <@ Enc.encrypt(d, m); (* ciphertext *)
    b <@ Adversary(Enc).detect_meek(c.sni_info, c.profile);
    return b;
  } }.
module Game0(Enc : TLS_ENCRYPTION) = {
  proc main() : bool = {
    d <@ Distr.background_domain_dist(); (* domain *)
    m <- cover_msg; (* message *)
    c <@ Enc.encrypt(d, m); (* ciphertext *)
    b <@ Adversary(Enc).detect_meek(c.sni_info, c.profile);
    return b;
  } }.
lemma true_positive_detects_normal:
  Pr[Game1(MeekEnc).main() = true] = 1%r.
lemma false_positive_from_poisson_tail:
  Pr[Game0(MeekEnc).main() = true] <= (1%r /% 10000%r) * (1%
      r /% 1000%r).
```

Summary of Implications. This model illustrates a realistic adversarial strategy that combines: (1) **SNI-based filtering**: blocks all traffic using the known front domain. (2) **Traffic pattern classification**: applies statistical profiling to reduce false positives.

Under the assumptions (*i*) meek traffic has consistent packet size behavior (normally distributed around 512) and (*ii*) background traffic does not resemble meek traffic and has distinct statistical characteristics. The implications are:

- the adversary achieves a **true positive rate of 1.0**, meaning it detects almost all actual meek flows;
- the **false positive rate is extremely low** due to the low overlap in SNI usage and distinct distributions; and
- therefore, meek is vulnerable to detection in environments where censors can combine visible metadata (SNI) with passive statistical analysis.

5 Challenges and Future Directions

This work represents an initial foray toward building a formal verification toolchain for analyzing censorship circumvention systems. Developing such a toolchain introduces a number of significant challenges across multiple dimensions, including protocol modeling, proof formalization, tooling limitations, and practical usability. At the modeling level, encoding HCS protocols within a simulation-based cryptographic framework demands a high-fidelity representation of adaptive network behavior and adversarial censorship strategies. Since many HCSs are designed to closely imitate benign traffic, formalizing these mimicry techniques in a way that supports rigorous analysis is both subtle and technically demanding.

From a proof perspective, simulation-based security requires constructing a simulator that can generate transcripts indistinguishable from those of the real system—often a nontrivial task, particularly when working with interactive protocols and adaptive adversaries. While EasyCrypt offers a powerful foundation for reasoning about probabilistic and relational properties, applying it to the domain of censorship resistance exposes limitations, especially as protocols become more interactive or as adversaries are modeled with advanced capabilities such as deep-packet inspection, traffic shaping, or selective packet dropping.

Finally, to support broader adoption within the circumvention research community, the long-term vision of this work involves designing a more accessible and user-friendly DSL. Such a DSL would lower the barrier to entry for protocol designers and researchers, enabling them to specify and verify HCS protocols without requiring deep expertise in formal methods. While the current effort is still in its early stages, it lays the groundwork for a principled and extensible framework for reasoning about the security of HCSs against powerful and evolving censorship adversaries.

6 Acknowledgments

This material is based on work supported by the Defense Advanced Research Project Agency (DARPA) under contract no. HR00112590081. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors, and do not necessarily reflect the views of the sponsors.

References

[1]]censorship-literature [n. d.]. CensorBib. https://censorbib.nymity.ch/. Accessed: 2025-04-21.

[2] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. 2019. SoK: Computer-Aided Cryptography. Cryptology ePrint Archive, Paper 2019/1393. https://eprint.iacr.org/2019/1393

[3] Diogo Barradas, Nuno Santos, Luís Rodrigues, and Vítor Nunes. 2020. Poking a hole in the wall: Efficient censorship-resistant Internet communications by parasitizing on WebRTC. In *CCS*.

[4] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. 2011. Computer-aided security proofs for the working cryptographer. In *Annual Cryptology Conference*. Springer.

[5] Kevin Bock, George Hughey, Xiao Qiang, and Dave Levin. 2019. Geneva: Evolving Censorship Evasion Strategies. In *ACM CCS*.

[6] Ankush Desai, Vivek Gupta, Ethan Jackson, Shaz Qadeer, Sriram Rajamani, and Damien Zufferey. 2013. P: safe asynchronous event-driven programming. In *Proceedings of ACM PLDI*.

[7] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. 2015. Blocking-resistant communication through domain fronting. *Privacy Enhancing Technologies* 2015, 2 (2015). https://www.icir.org/vern/papers/meek-PETS-2015.pdf

[8] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. 2013. The Parrot Is Dead: Observing Unobservable Network Communications. In *IEEE S&P*.

[9] Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Arjun Nitin Bhagoji, Paul Schmitt, Francesco Bronzino, and Nick Feamster. 2024. NetDiffusion: Network Data Augmentation Through Protocol-Constrained Traffic Generation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 1 (2024), 1–32.

[10] Michael Lack. 2024. PWND2: Provably Weird Network Deployment and Detection. https://www.darpa.mil/research/programs/provably-weird-network

[11] Yehuda Lindell. 2016. How To Simulate It - A Tutorial on the Simulation Proof Technique. Cryptology ePrint Archive, Paper 2016/046. https://eprint.iacr.org/2016/046

[12] Lauren Pick, Ankush Desai, and Aarti Gupta. 2023. Psym: Efficient Symbolic Exploration of Distributed Systems. *Proceedings of ACM PLDI* (2023).

[13] Marc B Rosen, James Parker, and Alex J Malozemoff. 2021. Balboa: Bobbing and weaving around network censorship. In *USENIX Security*.

[14] Zhen Sun and Vitaly Shmatikov. 2023. TELEPATH: A Minecraft-based Covert Communication System. In *S&P*.