Glimmers: Resolving the Privacy/Trust Quagmire

David Lie
University of Toronto/Google Brain

Petros Maniatis
Google Brain

ABSTRACT

Users today enjoy access to a wealth of services that rely on user-contributed data, such as recommendation services, prediction services, and services that help classify and interpret data. The quality of such services inescapably relies on trustworthy contributions from users. However, validating the trustworthiness of contributions may rely on privacy-sensitive contextual data about the user, such as a user's location or usage habits, creating a conflict between privacy and trust: users benefit from a higher-quality service that identifies and removes illegitimate user contributions, but, at the same time, they may be reluctant to let the service access their private information to achieve this high quality.

We argue that this conflict can be resolved with a pragmatic *Glimmer of Trust*, which allows services to validate user contributions in a trustworthy way without forfeiting user privacy. We describe how trustworthy hardware such as Intel's SGX can be used on the client-side—in contrast to much recent work exploring SGX in cloud services—to realize the Glimmer architecture, and demonstrate how this realization is able to resolve the tension between privacy and trust in a variety of cases.

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**; **Hardware-based security protocols**;

KEYWORDS

Privacy, SGX, Glimmers

ACM Reference format:
David Lie and Petros Maniatis. 2017. Glimmers: Resolving the Privacy/Trust Quagmire. In *Proceedings of HotOS '17, Whistler, BC, Canada, May 08-10, 2017,* 7 pages.
https://doi.org/10.1145/3102980.3102996

1 PRIVACY AND CLOUD SERVICES AT ODDS

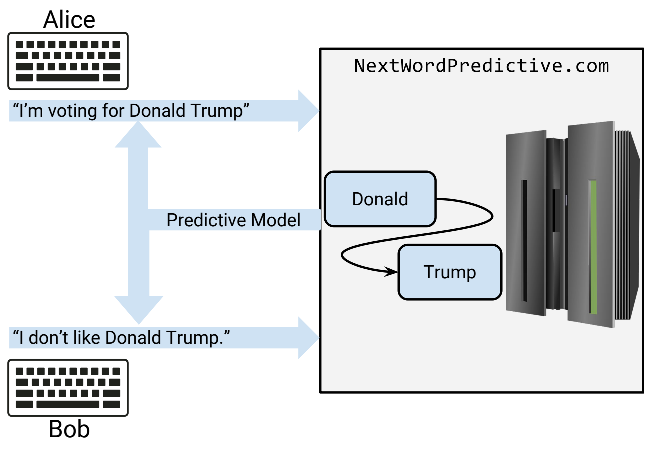
Many Internet services collect data contributed by users. User data contributions and aggregates gleaned from processing them are then often shared across all users of the service. For example, a predictive keyboard on a client device could benefit from a trained machine-learning model that uses inputs from different users' keyboards (Figure 1a). As current topics (such as "the world series" or "Donald Trump") trend up—because many users type them on their keyboards in a short time span—an up-to-date model can suggest "Trump" as the next word when Alice types "Donald", even if she has never typed that name herself before. Similar benefits exist for a wide range of applications, such as image recognition, recommendation systems, and activity detection.

Sadly, sharing of information, especially deeply personal information such as key clicks, naturally introduces a tussle between information utility and user privacy. If Alice's keyboard streams all her key clicks to a shared, predictive keyboard service, she will almost certainly reveal sensitive information about her political beliefs. Although they can meaningfully contribute to learning a global predictive word model, Alice's individual key clicks *are not* public information, and permitting their use for training *is not* tantamount to publishing them on a world-readable web page. Yet most service providers release scant, if any, information about what precisely they do with data they collect, which can be troubling for users who both want to help but are also concerned about privacy. For example, Bob's disparaging remarks in Figure 1a can enable a malicious service provider to discriminate against him or aid in his prosecution. Even if the service provider is well-meaning, it could be compelled by a court order to release this information to the government to persecute Bob, stolen by hackers to extort him or damage the service brand, or used by regulatory bodies to slam the service provider with privacy-violation fines. Therefore, unmediated information sharing is undesirable to users and services alike.

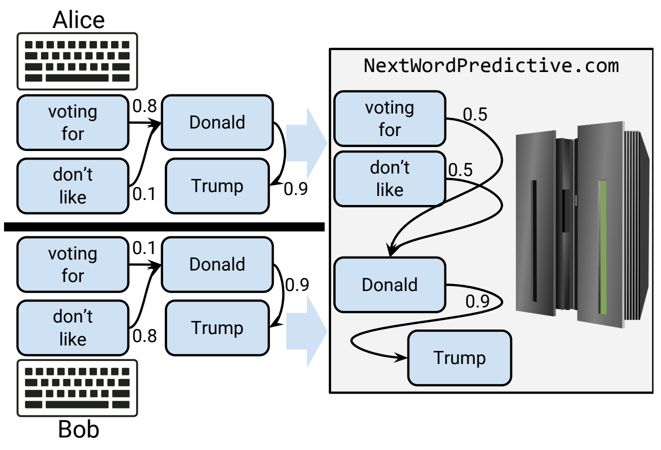
One way to alleviate this tussle is to process user data at the client, so that it is never disclosed to the service in the raw [12, 16]. With Federated Machine Learning [12], every client computes a local partial model, and the service sums those models together to generate a global one. For example, a simplistic keyboard model in Figure 1b associates a weight between 0 and 1 for an ordered pair of words, without revealing the actual sentences typed. However, learned models, even ones much more sophisticated than our straw-man illustration, can still reveal information about the raw inputs used to train those models (e.g., machine-learning models can be inverted [6]).

Consequently it behooves users to reduce the amount of their private information the service has access to, whether pre-processed at the client or not. Note, furthermore, that many services need no access to the contributions of any individual, but only to aggregates over a large population of users; for such services, receiving and storing the private data of individual users brings unnecessary cost, risk, and liability. Therefore, and satisfying both sides of the divide,

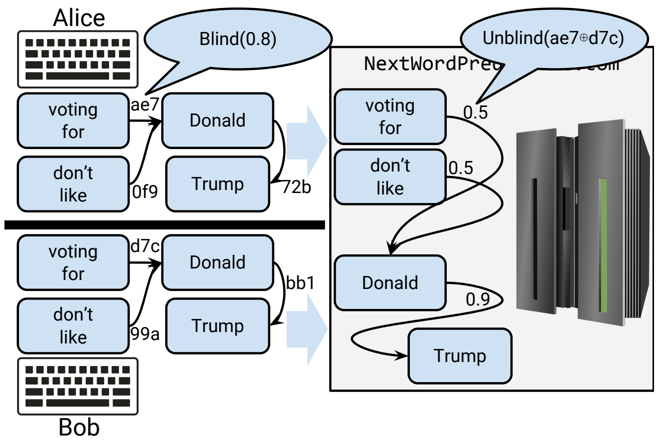
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
HotOS '17, May 08-10, 2017, Whistler, BC, Canada
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5068-6/17/05.
https://doi.org/10.1145/3102980.3102996



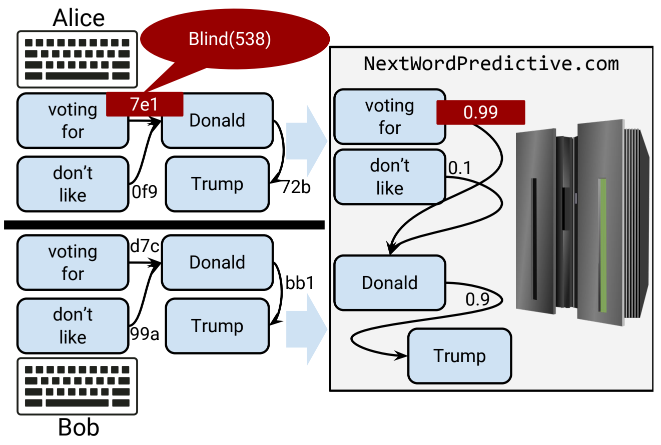
(a) A next-word predictive service for users' keyboards.



(b) Federated machine learning.



(c) Secure model aggregation.



(d) Attack against secure model aggregation.

Figure 1: A predictive keyboard service

the service could have users contribute information using cryptographic blinding [4] (Figure 1c), or similar techniques that enable accurate aggregation of a data set while still hiding the individual data items contributed.

Unfortunately, such techniques hide the raw inputs from the service and, consequently, afford malicious users the opportunity to contribute illegitimate inputs undetected, derailing the quality of the shared service. For example, Alice could contribute a blinded local model for her own keyboard sequences that has been maliciously manipulated to over-weight her personal political convictions (i.e., contributing an illegal value of 538 for one model parameter, violating the valid rage of $[0, 1]$), making it seem extremely popular beyond what a single user might make it (Figure 1d). When the service aggregates the blinded local models together, it cannot detect such induced bias, even if specific validation checks such as range checks existed, because of the blinding. As a result, the service ends up with a catastrophically skewed global predictive model towards inaccurate predictions, degrading the experience of all users.

Such attacks are not of marginal import and should be taken seriously: changing the order of predictions might skew thought processes, as has been shown for search-rank manipulation affecting election results [5]. In our example, the obvious way to check Alice's contribution is before blinding occurs, but the service cannot trust Alice to do that faithfully, since she is the adversary in that threat. Even if the actual user contributions are not themselves private, e.g., users' photos associated with a location on a mapping service, validating those contributions might require access by the service to otherwise private data (e.g., location history through GPS and ambient WiFi, to validate that the user did go to a claimed location) on the device of the very user whose malice the validation is meant to protect from.

Services affected by this trust-privacy trade-off have similar characteristics: a) they consume user-contributed data to build global state benefiting all users; b) service quality is highly dependent on the trustworthiness of data contributed by users; and c) they can only verify the legitimacy of user contributions through direct access to sensitive user data (the contributions themselves, or contextual user information such

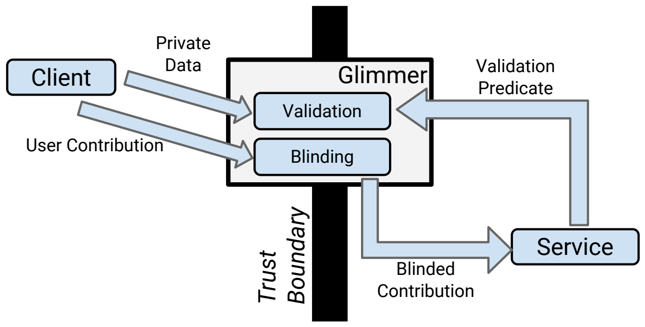


Figure 2: Glimmer architecture.

as logs and other user activity). Many popular services have these properties. For example, image recognition can benefit from local partial models trained on the private photos of users, but to verify that contributed photos are legitimate requires direct access to them before they are blinded; recommender services learn similarities among products from individual users' registered likes, dislikes, and shopping habits, but detecting spurious reviews requires access to individual users' purchasing history; activity-recognition models improve from analyzing silhouettes and image structure from in-home cameras, but checking that silhouettes are legitimate requires analysis of full video streams captured at people's homes. Given the state of things, one may sadly conclude that trust and privacy in such services is a zero-sum game—an increase in individual privacy results directly in a decrease in the amount of trust all the users can collectively place in the service. In the next section, we introduce *Glimmers* to exit this quagmire.

2 A GLIMMER OF TRUST IN BETWEEN

To visualize the conundrum between privacy and trust, we introduce the concept of a *Trust Boundary* between the client and the service; no private data should cross from the client to the service, but the correctness of client contributions must be checked according to criteria set by the service on the client's side. The server cannot establish trust without access to the private data, but at the same time the client does not want the server to access its private data unrestricted.

Such "air-gap" problems can be solved with the introduction of a trusted third party that performs *validation* on private client data before submitting user contributions to the service. We call our logical trusted third party a *Glimmer of Trust*, or Glimmer for short, since it performs very limited but essential trusted functionality: validation of private data as specified by the service, followed by submission to the service (Figure 2).

We use the term validation loosely here to capture any validity predicate delegated to the trusted third party; different validation predicates may trade off computational complexity for result accuracy. For example, in the next-word prediction service, range-checking model parameters ensures that Alice cannot send a user contribution of 538 when a value between 0 and 1 is expected; however, she can still send arbitrary fictitious values within that range that may not correspond to her actual keyboard activity. A more sophisticated validator might instead observe actual keyboard behavior (a la NAB [7]) to match keyboard events to reported model weights; or even observe CPU branches [20] to identify a plausible execution of the model-construction code that produced contributed partial results, as has been suggested for on-line game cheat detection [3]. While more invasive validation increases the complexity and resources required by the Glimmer, it also increases the adversary's cost to cheat undetected, since she now has to fabricate keyboard activity or program executions that corroborate her deceptive inputs to the service.

Regardless of the actual validation semantics, the Glimmer must satisfy certain properties to be helpful. First, it must guarantee that it either discards private inputs after processing, or that it blinds them if the private data is part of the contribution, thus bounding the amount of information about private data that is leaked (**Input Confidentiality**). Second, it only endorses for use by the service those contributions that it has validated (**Input Integrity**).

Having an actual third party performing the role of the Glimmer is, arguably, the realization of this architecture. For example, the Electronic Frontier Foundation (EFF), or a consortium of privacy-advocacy organizations could, in ensemble, perform validation and blinding, perhaps using multi-party computation, or simpler threshold cryptography on inputs. However, the deployment cost for such a solution would be high. In this vision paper, we focus on using trustworthy hardware, because of its current broad availability, as an implementation platform for Glimmers performing privacy-preserving user-data validation. Our work revisits the early vision of using secure computing elements as client-local trusted third parties [1], in particular within the context of protecting user privacy.

3 GLIMMERS ON SGX

There are several instances of trustworthy hardware commonly available on computing clients today, such as Intel TXT, AMD SVM, ARM TrustZone, and Intel SGX. In general, all these platforms provide a hardware-enforced trusted execution environment (TEE), which can execute functionality isolated from any vulnerabilities or malicious code.

In this paper we focus on realizing Glimmers using Intel's SGX [9]. SGX has spawned renewed interest in trusted computing, with a number of server-side uses [2, 8, 15]. We are instead studying how SGX can be used on clients to realize Glimmers of Trust—note that, at the time of writing, SGX is only available on client-class CPUs. SGX provides a TEE called an *enclave*. In addition to isolation, an SGX enclave also supports *remote attestation*, which allows it to prove cryptographically to a remote party that it is running correctly in a legitimate enclave. Finally, it provides *sealed storage*, which allows it to encrypt data so that only it or other designated binaries, running in a legitimate enclave, can

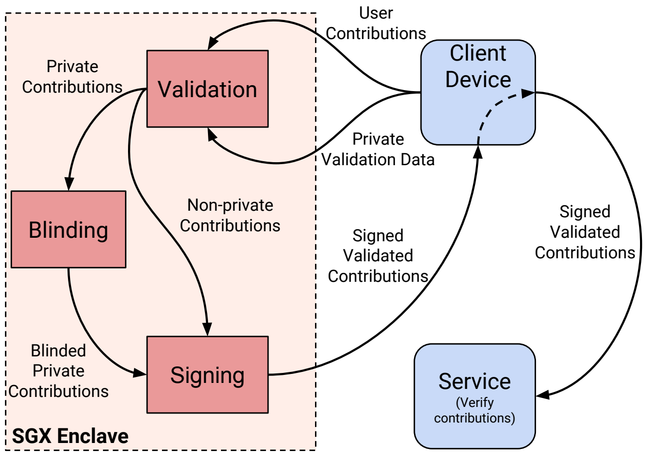


Figure 3: SGX Glimmer design

decrypt it. Although very powerful, SGX enclaves operate using limited resources, have no direct access to privileged CPU operations such as IO, and must mediate system services via the untrusted host OS. As a result, Glimmers implemented as enclaves must be simple and run mostly in isolation.

Figure 3 describes the design of an SGX-based Glimmer. The Glimmer has 3 main components. A Validation component takes two types of input from the client device: user contributions, which will be sent to the service, and private validation data, which is used internally by the Glimmer to run the validation predicate. In some cases, user contributions are used in the validation predicate directly, while in other cases separate contextual private information is used there instead. For example, in the predictive keyboard service, model parameters are range-checked against the valid weight range $[0, 1]$. In contrast, more invasive validation predicates could request additional data, not contributed to the service, such as individual key presses and timings or web browser logs showing the typed data in user-triggered HTTP GET requests to corroborate the user contributions. For an application that attaches photographs of places to locations on a map, the Glimmer could request validation information such as exact GPS location and tracks, a fingerprint of the camera hardware, and access to other photos on the device to establish context. Note that the Glimmer cannot directly obtain such information; it must request it from the host system.

Blinding is the second Glimmer component. Its purpose is to hide private, user-contributed values so that the service can compute aggregates on them without revealing individual contributed data. Blinding must be performed together with Validation, since validation is impossible after blinding has occurred. To illustrate how this could work, we give a simple example [4]. Assume the existence of a trusted blinding service—which could, itself, be implemented as a separate enclave on one of the clients, or as a distinct trusted service—that computes N random blinding values p_i such that $\sum_{i=0}^{N-1} p_i = 0$. It then seals each p_i value to the Glimmer code, and encrypts one of the sealed values to each of N clients' public keys, distributing the encrypted blinding values to the Glimmers running on each client. Each Glimmer for user i can decrypt and unseal its blinding value. The Blinding component then computes the blinded user contribution $y_i = x_i + p_i$. y_i is safe to send to the service, since the service cannot compute the private x_i from it (because the blinding value p_i is secret). However, once the service sums all y_i's together, it can compute the accurate sum of all x_i's: $\sum_{i=0}^{N-1} y_i = \sum_{i=0}^{N-1} x_i + \sum_{i=0}^{N-1} p_i = \sum_{i=0}^{N-1} x_i$. Recall that non-private user contributions need not be blinded; for instance, in the crowd-sourced photos for map locations, user-contributed photos are meant to be shared, so they do not need to be blinded.

The third Glimmer component, Signing, takes a user-contributed input (blinded or unblinded) and the result of the Validation component, which can be a boolean "valid" / "invalid", or a confidence value. If validation passed, the Signing component signs the user-contributed input and returns it to the client for transmission to the service. To close any side channels, if validation fails, the Glimmer may still sign the contribution, but that contribution would contain a bit indicating that it is invalid, which only the service should be able to interpret. The signing key used can be provided by the service, and sealed (using the SGX sealing facility) to the Glimmer code, so that it is only available to instances of Glimmer enclaves.

One last requirement is that the Glimmer convince both the user and service that it is correct—i.e., that it has both input confidentiality and integrity properties. To convince the user, we envision vetting and formal verification by a third-party, such as the EFF; while the service could perform its own vetting and verification to convince itself. Once it has been vetted, the hash of the Glimmer is published, and the user can use SGX to attest that their client is running the approved Glimmer. Similarly the service can ensure that signing keys are sealed to the approved Glimmer. Because the Glimmer is, necessarily, small and limited in its external interactions, it is amenable to formal verification for absence of runtime errors such as buffer and integer overflows [11, 19]. Furthermore, much research has been recently devoted to verifying formally the confidentiality of secret values in SGX enclaves [17, 18]. The burden on programmers is relatively low: programming in a simple programming language (e.g., C) with relatively low-complexity idioms (e.g., bounded loops, no function pointers, etc.), explicitly marking secret inputs, explicitly marking declassification functions (e.g., blinding and encryption). Simple functional property verification can establish that every signed value has been validated and that no private information leaves the Glimmer without being blinded.

We have shown all components in Figure 3 within a single SGX enclave, which is more efficient as there is only one transition in and out of the enclave. However, to increase ease of verification, the Glimmer can be decomposed so that each component runs in its own enclave. Naturally, communication between components must now also be secured.

4 DEPLOYMENT CONSIDERATIONS

In this section we consider how Glimmers can be applied in challenging deployment scenarios, either with sensitive validation predicates, or when SGX is not readily available at the client devices needing Glimmer functionality.

4.1 Validation confidentiality

So far, we have described the use of Glimmers in applications where user contributions are used for a shared service. However, Glimmers have applications to other problems where privacy and trust are at odds. For example, consider the case of bot detection in a web service. While CAPTCHAs are a standard method for detecting bots, they have their drawbacks, such as vulnerability to computer vision and CAPTCHA farms, and annoyance to legitimate human users. An alternative solution is embedding a JavaScript "detector" in the web page that heuristically detects whether a human—as opposed to a bot—is present. Such solutions collect a large set of signals, such as how faithfully the client executes JavaScript, fingerprints of the client's system software and hardware, and the timing and frequency of UI interactions such as mouse movements and changes in focus [7, 14]. These signals are sent back to the web service, which uses them to determine if the sender is a bot or a human. However, these signals often contain private information, such as the user's cookies, browsing history, and browsing interests [10]. A Glimmer can protect individual privacy by performing the validation, which requires access to sensitive information locally on the client machine, and sending only 1 bit of information—whether the user is human or not—back to the web service.

In such an adversarial example, the web service may wish to hide the exact validation predicate from the adversary, a property we call **Validation Confidentiality**. Glimmers can provide validation confidentiality by accepting encrypted code and data from the web service and decrypting and running that code inside the enclave where the plain text code is protected from observation by the hardware TEE.

One challenge is to make sure that the keys used to sign and encrypt the code and data are transferred securely to the Glimmer and that the Glimmer only accepts keys from a legitimate web service. This can be accomplished using remote attestation, which enables data, such as Diffie-Hellman (DH) handshake values, to be bound to code running in an enclave. This would assert to the service that the DH handshake is occurring with a legitimate Glimmer. Similarly, the Glimmer would need to ensure that the DH handshake is occurring with a legitimate service, which can be accomplished by the service signing its DH handshake values and embedding the signature verification key in the Glimmer code. Once shared secrets are negotiated with DH key exchange, secret code and data can be securely transferred from the service to the Glimmer.

The other challenge is to prove input confidentiality to the user when part of the Glimmer can no longer be audited because it is encrypted and set dynamically at runtime. This can be done by making the message format between the Glimmer and the service public, and having a runtime auditor check that each message is well formed and contains only one bit of information (i.e., a single bit plus a well-defined signature and challenge response). While this does not preclude a covert channel, it puts a hard upper bound on the capacity of such a channel.

4.2 Glimmer-as-a-Service

So far we have proposed that Glimmers run on client devices. However, given the increasing trend towards Internet of things (IoT) devices, there are likely to be some devices that will make user contributions that must be trustworthy, but do not have a processor with trusted computing capabilities. In this case, we envision that a neutral—but not necessarily *trusted*—third party may supply the capability to run a Glimmer as a service, on behalf of clients. This third party could simply be another device owned by the same user (such as a set-top box or home service), a local group of people that the user knows (such as their University, community, or church), or even a well-known entity that is willing to sell or provide resources to improve user privacy. Note that unlike trusted third parties, a Glimmer service still relies on SGX to provide its guarantees, and need not necessarily be trusted organizationally.

The main criterion is that the client device needs to establish that it is sending its private data to a genuine Glimmer. Fortunately, this can be accomplished using the same attestation mechanism to establish a secure channel as described above. Attestation enables the client to be assured that the other endpoint of a secure communication channel is within an approved Glimmer. Once this is done, the client can transmit the user contribution and private data and receive in return a signed (and if necessary, blinded) user contribution, which it can then forward to the service.

5 CHALLENGES AND FUTURE WORK

The effectiveness of a Glimmer depends on the effectiveness and practicality of the validation predicate used to decide whether a user contribution is trustworthy. Without a trusted path to the point of collection for user-contributed data, the validation function must be resistant to data forgery. Thus, the validation problem consists of a) checking the consistency of multiple, untrusted sources of data to detect the existence of forged contributions, and b) checking the contributed data against a model describing typical, trustworthy contributions. As a result, the success of validation depends on the secrecy (i.e., validation confidentiality) of some aspects of the predicate: either the exact data it collects, or the validation model it uses. Fortunately, TEEs are designed to protect the confidentiality and integrity of intermediate values of functions they execute; even so, functions that are hard or impossible to reverse-engineer by observing their inputs and outputs are an active, open research problem. Some initial thoughts for our future work include collecting much more validation data than the model requires, hiding or perturbing randomly the outcome of validation, and obfuscating the validation predicate itself.

In addition, TEEs such as SGX and Trustzone have restrictions on what types of functionality they support; for example, SGX enclaves can only invoke user-space, ring-3 instructions. As a result, they cannot directly take system-level observations about inputs and outputs to devices, device drivers, buffer caches, browser histories, etc. This has been a persistent problem with TEEs even from prior generations. Although some research proposals on establishing trusted path have been published [21], no pervasive solutions exist. This obstacle can be partially surmounted through the use of external observations that are less amenable to tampering (e.g., by middleboxes or trusted external services observing and recording a client's behavior). Another promising approach may involve the attested communication of a relatively performant but user-space-only TEE (e.g., an SGX enclave) with a slower-to-use, but system-level capable TEE (e.g., a verified mini hypervisor booted via Intel's TXT [19], or a late-launched TEE [13]); the latter can collect trustworthy system observations, which it can periodically attest to, for the use of the former's frequent validation checks. Our future work is studying how such approaches can improve the quality of validation checks for Glimmer's use cases.

6 CONCLUSION

We propose Glimmers of Trust, implemented on trusted computing hardware, that can provide trustworthiness guarantees of user-contributed data to services without compromising user privacy. We describe a design using Intel SGX, which we are currently implementing. While many previous proposals for SGX are for server-side uses [2, 8, 15], or confer mainly server-side benefits (i.e., DRM, mobile payments), we see Glimmers as one of the first uses of client-side trusted computing that can benefit *both* services and users.

ACKNOWLEDGMENTS

The ideas presented in this paper were inspired by a number of discussions with Úlfar Erlingsson. Keith Bonawitz, Arnar Birgisson, and Martín Abadi gave us valuable feedback on earlier drafts of this document.

REFERENCES

[1] Martín Abadi. 2004. Trusted Computing, Trusted Third Parties, and Verified Communications. In *Security and Protection in Information Processing Systems*. Springer US, Toulouse, France, 291–308. https://doi.org/10.1007/1-4020-8143-X_19

[2] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan OtextquoterightKeeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rdiger Kapitza, Peter Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (2016). USENIX Association, 689–703. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov bibtex: 199364.

[3] Darrell Bethea, Robert A. Cochran, and Michael K. Reiter. 2008. Server-side Verification of Client Behavior in Online Games. *ACM Trans. Inf. Syst. Secur.* 14, 4, Article 32 (Dec. 2008), 27 pages. https://doi.org/10.1145/2043628.2043633

[4] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2016. Practical Secure Aggregation for Federated Learning on User-Held Data. In *Proceedings of the 2016 Workshop on Private Multi-party Machine Learning (PMPML'16)*.

[5] Robert Epstein and Ronald E. Robertson. 2015. The Search Engine Manipulation Effect (SEME) and its Possible Impact on the Outcomes of Elections. *Proceedings of the National Academy of Sciences* 112, 33 (2015), E4512–E4521. https://doi.org/10.1073/pnas.1419828112 arXiv:http://www.pnas.org/content/112/33/E4512.full.pdf

[6] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, New York, NY, USA, 1322–1333. https://doi.org/10.1145/2810103.2813677

[7] Ramakrishna Gummadi, Hari Balakrishnan, Petros Maniatis, and Sylvia Ratnasamy. 2009. Not-a-Bot: Improving Service Availability in the Face of Botnet Attacks. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'09)*. USENIX Association, Berkeley, CA, USA, 307–320. http://dl.acm.org/citation.cfm?id=1558977.1558998

[8] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. 2016. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (2016). USENIX Association, 533–549. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/hunt bibtex: 199358.

[9] Intel 2014. Intel® Software Guard Extensions Programming Reference. (Oct. 2014). Available at https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.

[10] Dongseok Jang, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. 2010. An Empirical Study of Privacy-Violating Information Flows in JavaScript Web Applications. In *Proceedings of CCS 2010*, Angelos Keromytis and Vitaly Shmatikov (Eds.). ACM Press, 270–83.

[11] Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. 2015. Frama-C: A Software Analysis Perspective. *Formal Aspects of Computing* 27, 3 (2015), 573–609. https://doi.org/10.1007/s00165-014-0326-7

[12] Jakub Konecný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. 2015. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. In *Proceedings of the NIPS Workshop on Optimization for Machine Learning*.

[13] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. 2010. TrustVisor: Efficient TCB Reduction and Attestation. In *Proceedings of the IEEE Symposium on Security and Privacy*.

[14] KyoungSoo Park, Vivek S. Pai, Kang-Won Lee, and Seraphin B. Calo. 2006. Securing Web Service by Automatic Robot Detection. In *USENIX Annual Technical Conference, General Track* (2006). 255–260. http://static.usenix.org/event/usenix06/tech/full_papers/park/park_html/

[15] Felix Schuster, Manuel Costa, Cdric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *Security and Privacy (SP), 2015 IEEE Symposium on* (2015). IEEE, 38–54. http://ieeexplore.ieee.org/abstract/document/7163017/

[16] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-Preserving Deep Learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1310–1321.

[17] Rohit Sinha, Manuel Costa, Akash Lal, Nuno P. Lopes, Sriram Rajamani, Sanjit A. Seshia, and Kapil Vaswani. 2016. A Design and Verification Methodology for Secure Isolated Regions. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '16)*. ACM, New York, NY, USA, 665–681. https://doi.org/10.1145/2908080.2908113

[18] Rohit Sinha, Sriram Rajamani, Sanjit Seshia, and Kapil Vaswani. 2015. Moat: Verifying Confidentiality of Enclave Programs. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, New York, NY, USA, 1169–1184. https://doi.org/10.1145/2810103.2813608

[19] Amit Vasudevan, Sagar Chaki, Petros Maniatis, Limin Jia, and Anupam Datta. 2016. überSpark: Enforcing Verifiable Object Abstractions for Automated Compositional Security Analysis of a Hypervisor. In *USENIX Security Symposium*. USENIX Association, Austin, TX, 87–104. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/vasudevan

[20] Amit Vasudevan, Ning Qu, and Adrian Perrig. 2011. XTrec: Secure Real-Time Execution Trace Recording on Commodity Platforms. In *System Sciences (HICSS), 2011 44th Hawaii International Conference on*. IEEE, 1–10.

[21] Zongwei Zhou, Virgil D. Gligor, James Newsome, and Jonathan M. McCune. 2012. Building Verifiable Trusted Path on Commodity x86 Computers. In *Proceedings of the IEEE Symposium on Security and Privacy*.