VICEROY: GDPR-/CCPA-compliant Enforcement of Verifiable Accountless Consumer Requests

Scott Jordan
UC Irvine
sjordan@uci.edu

Yoshimichi Nakatsuka
UC Irvine
nakatsuy@uci.edu

Ercan Ozturk
UC Irvine
ercano@uci.edu

Andrew Paverd
Microsoft Research
andrew.paverd@microsoft.com

Gene Tsudik
UC Irvine
gene.tsudik@uci.edu

***Abstract*—Recent data protection regulations (notably, GDPR and CCPA) grant consumers various rights, including the right to *access*, *modify* or *delete* any personal information collected about them (and retained) by a service provider. To exercise these rights, one must submit a *verifiable consumer request* proving that the collected data indeed pertains to them. This action is straightforward for consumers with active accounts with a service provider at the time of data collection, since they can use standard (e.g., password-based) means of authentication to validate their requests. However, a major conundrum arises from the need to support consumers *without accounts* to exercise their rights. To this end, some service providers began requiring such *accountless* consumers to reveal and prove their identities (e.g., using government-issued documents, utility bills, or credit card numbers) as part of issuing a verifiable consumer request. While understandable and reasonable as a short-term fix, this approach is cumbersome and expensive for service providers as well as privacy-invasive for consumers.**

Consequently, there is a strong need to provide better means of authenticating requests from accountless consumers. To achieve this, we propose `VICEROY`, a privacy-preserving and scalable framework for producing *proofs of data ownership*, which form a basis for verifiable consumer requests. Building upon existing web techniques and features, `VICEROY` allows accountless consumers to interact with service providers, and later prove that they are the same person in a privacy-preserving manner, while requiring minimal changes for both parties. We design and implement `VICEROY` with emphasis on security/privacy, deployability and usability. We also thoroughly assess its practicality via extensive experiments.

I. Introduction

Several new data protection regulations have been enacted in recent years, notably the European Union General Data Protection Regulation (GDPR) [62] and the California Consumer Privacy Act (CCPA) [58]. These regulations grant consumers various new legal rights[1]. For example, consumers gain the right to *access* personal data collected about them and held by service providers (GDPR Art. 15, CCPA 1798.100), *request correction* (GDPR Art. 16, CCPA 1798.106) or *request deletion* of their personal data (GDPR Art. 17, CCPA 1798.105).

Importantly, these regulations expand the definition of *personal data* beyond that associated with a person's real name. For example, GDPR Rec. 30 states that natural persons "*may be associated with online identifiers provided by their devices, applications, tools and protocols, such as internet protocol addresses, cookie identifiers or other identifiers*". This means that any website[2] collecting information about consumers based on identifiers, such as IP addresses or cookies, may be collecting personal information, and thus have to comply with these new regulations. The website must therefore provide a means by which consumers can access, request correction of, or request deletion of their personal information.

When dealing with a consumer request, the website must verify that the requestor is indeed the consumer to whom the personal information pertains. This is critical to prevent erroneous disclosure (which would be a serious violation of any data protection regulation), unauthorized modification, or deletion of personal information. This is called a "*verifiable consumer request*" (VCR) (CCPA 1798.140(y)).[3]

For consumers who have pre-existing accounts on a given website, submitting a VCR is relatively straight-forward. To wit, CCPA 1798.185 requires: "*treating a request submitted through a password-protected account maintained by the consumer ... as a verifiable consumer request*". However, there remains a major challenge of how to support a VCR from casual or **accountless** consumers, as required by CCPA 1798.185, whilst protecting such consumers' privacy.

The only mechanisms that are currently suitable for accountless consumers are those that require: a device cookie, a government-issued ID, a signed (and possibly witnessed) statement, a utility bill, a credit card number, or taking part in a phone interview [64]. However, these mechanisms are cumbersome (and some are time-consuming) for consumers as well as insecure, as demonstrated by prior work [64], [40], [47], [39]. They typically require manual processing, which is both error-prone and costly. Moreover, such methods (apart from device cookies) are privacy-invasive for the consumer and open the door for further consumer data exposure. For example, a government-issued ID or utility bill reveals even more private information to the website.

In light of these issues, the most appealing choice appears

[1] We use the term *consumer* to refer to: (1) the GDPR term *data subject*, (2) the CCPA term *consumer*, and (3) the equivalent terms in other regulations.

[2] As shorthand, we use the term *website* to represent the entity operating a website, which (we assume) falls into the category of entities that the GDPR and CCPA call *controller* and *business*, respectively.

[3] These requests are sometimes also referred to as Subject Rights Request (SRR) or Subject Access Request (SAR).

Network and Distributed System Security (NDSS) Symposium 2023
28 February - 4 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.23074
www.ndss-symposium.org

to be the use of device cookies. Cookies are already used pervasively by websites to link multiple sets of activities (sessions) to the same consumer. At first glance, asking for device cookies as part of a VCR appears to meet the GDPR and CCPA requirements: only the authorized consumer should possess the correct cookie (*unforgeability*), and providing a cookie does not reveal additional information (*privacy*). However, this essentially means treating device cookies as *authentication tokens*, which has at least three disadvantages:

First, in the general case, there is no requirement for a cookie's value to be unguessable. A recent large-scale study [52] found cookie values containing URLs, email addresses, timestamps, and even JSON objects. Although *authentication cookies* are designed to be unguessable, these would typically only be used once the user has logged into an account.

Second, cookies are used (i.e., sent over the network) whenever the consumer interacts with the website. Although secure communication channels (e.g., TLS) can protect cookies in transit, the MITRE ATT&CK framework lists several recent examples of techniques for stealing web cookies [19].

Third, consumers must protect the cookies stored on their devices, especially considering e.g., client-side spyware, even after the cookies expire. Secure storage may become more challenging as the number of stored cookies increases, since consumers should not delete cookies for which they might subsequently issue a VCR. If an adversary can guess or obtain the cookie through any of the above vectors, they would be able to request, modify, or delete all the consumer's data.

Motivated by aforementioned issues, we construct VICEROY, a first-of-its-kind framework that allows *accountless* consumers to request their data in a private manner, while allowing website operators to efficiently and securely verify such requests. VICEROY introduces a one-to-one mapping between Web sessions and consumer-generated public keys. At session initiation, the consumer generates a public key (a VCR public key) and supplies it to the server. At a later time, the consumer digitally signs its request using the private key corresponding to the VCR public key for the session.

To ensure consumer privacy, our key derivation mechanism uses unlinkable public keys derived from a single master public key. This also allows VICEROY to only require consumers to securely store a single private key, regardless of the number of sessions they have generated. Moreover, since this private key is only needed when generating VCRs, it can be protected using well-known secure key storage mechanisms (e.g., hardware security device).

VICEROY is composed of well-known cryptographic primitives. However, to meet the necessary requirements of security, scalability, and privacy, this must done through the careful selection of such primitives. Furthermore, VICEROY's design prioritizes deployablity, requiring only minimal changes to existing websites and no changes to existing cookie usage.

The contributions of this work are:

- Design of VICEROY– a secure, scalable, and privacy-preserving VCR mechanism for accountless consumers.
- Careful selection of cryptographic protocols to balance the three requirements of VICEROY.
- Proof-of-concept implementation of VICEROY for web browsers, including a VICEROY-compatible hardware security device.
- Thorough security, performance, and deployability evaluation of VICEROY.

Organization: Section II presents background on GDPR/CCPA and verifiable consumer requests (VCRs). Next, Section III presents our threat model and defines requirements for VICEROY. Sections IV and V then describe the design and proof-of-concept implementation of VICEROY. Section VI presents our evaluation methodology and results. Further aspects of VICEROY are discussed in Section VII and related work is overviewed in Section VIII. Section IX concludes the paper.

Code Availability. Source code for all VICEROY components and the Tamarin model is available at [2].

II. GDPR/CCPA Background

This section overviews Personally Identifiable Information and consumer rights under the GDPR and the CCPA. Given familiarity with GDPR and CCPA, it can be skipped without any loss of continuity.

A. Personally Identifiable Information (PII)

Both GDPR and CCPA pertain to the combination of *personal and personally identifiable* information, often referred to as: *Personally Identifiable Information*[4] or PII.

Information is *personal* if it relates to a person, e.g., contact information, geolocation, applications and devices used, how an application is used, interests, websites visited, consumer-generated content, identities of people with whom a consumer communicated, content of communication, audio, video, and sensor data [56].

Information is *personally identifiable* if the person to which it pertains is either identified or identifiable. If information is paired with a name, telephone number, email address, government-issued identifier, or postal address, then it is considered to be personally identifiable [55]. If information is paired with an IP address, a device identifier (e.g., an IMEI), or an advertising identifier, it is *likely* to be considered as personally identifiable [55]. Information paired with an identifier created by a business (e.g., a cookie) is personally identifiable if it can be combined with other information to allow the consumer to whom it relates to be identified [55].

B. Rights of Access and Erasure

Both GDPR and CCPA require a business that collects PII to disclose, typically in its privacy policy, the categories of PII collected, the purposes for collecting it, and the categories of entities with which that PII is shared [55]. Both regulations give consumers the right:

- To learn about, and control, information relating to them that a business has collected. Specifically, consumers have the right to request access to the *specific pieces of PII* that

[4]The GDPR uses the term *personal data* and the CCPA uses *personal information.*

the business has collected (GDPR Art. 15; CCPA Sec. 1798.110(a)(5)).
- To request that their incorrect PII be corrected (GDPR Art. 16; CCPA Sec. 1798.106).
- To request that a business delete their PII (GDPR Art. 17; CCPA Sec. 1798.105).

C. Verifiable Consumer Requests (VCRs)

The consumer rights to access, and request correction or deletion of, their PII are contingent upon verification that the consumer is indeed the person to whom that PII relates. However, GDPR and CCPA differ in requirements of methods of verification. Both regulations require a business to use reasonable measures to verify the consumer's identity (GDPR Rec. 64; CCPA Sec. 1798.140(ak)). If a consumer has a password-protected account with a business, both require a business to treat requests submitted via that account as verified (GDPR Rec. 57; CCPA Sec. 1798.185(a)(7)).

However, both regulations also recognize that PII is often collected about casual consumers, who do not have password-protected accounts. In this case, they envision a consumer request being verified by associating additional consumer-supplied information with PII that the business previously collected about that consumer (CCPA Sec. 1798.130(a)(3)(B)(i)). The CCPA further specifies that any information provided by the consumer in the request can be used solely for the purposes of verification (CCPA Sec. 1798.130(a)(7)). However, if a business has not linked PII to a consumer or a household, and cannot link it without the acquisition of additional information, then neither the GDPR nor the CCPA require a business to acquire additional information to verify a consumer request (GDPR Rec. 57; CCPA Sec. 1798.145(j)(3)). Thus, some requests may be unverifiable.

The CCPA [51] recognizes that consumer verification is not absolutely certain. It establishes two thresholds of certainty. The lower threshold, called *reasonable degree of certainty*, may be satisfied by matching at least two pieces of information provided by the consumer (CCPA Regs. §999.325(b)). The higher threshold, called *reasonably high degree of certainty*, may be satisfied by matching at least three pieces of information provided by the consumer, and obtaining a signed declaration from the consumer (CCPA Regs. §999.325(c)). However, other means of verification may also satisfy these thresholds. Verification of the consumer identity must always, at a minimum, meet the *reasonable degree of certainty* threshold. Furthermore, requests to learn specific pieces of PII must meet the *reasonably high degree of certainty threshold*. Finally, a consumer may choose to use a third-party verification service (CCPA Regs. §999.326).

The design of a verification method should balance the administrative burden on the consumer (CCPA Sec. 1798.185(a)(7)) with the likelihood of unauthorized access and the risk of harm (CCPA Regs. §999.323(b)(3)).

III. Threat Model and Requirements

Our system model assumes a typical Web environment with two types of principals: *(1)* clients and *(2)* servers. Clients are *consumers* who access Internet services offered by servers. Servers collect and store data during the interactions with the consumers by associating such data with identifiers issued to the consumer. Each client can own multiple devices and at least one of the client's devices can be trusted to store a secret, e.g., a private key. This trusted device could be a smartphone, a dedicated key storage device, or a secure hardware wallet. All access to the secret is controlled by the client. Physical and side-channel attacks against the trusted client device are beyond the scope of this paper.

We assume secure communication channels between clients and servers, which can be realized using standard means, e.g., HTTPS. Use of secure channels to deliver web content has become a de-facto standard, as shown by Felt et al. [49], which reports that up to 87% of all webpages were served via HTTPS in 2017. This number is expected to increase, as shown by Google's 2022 Transparency Report [14], which claims that 80–98% of top-100 websites use HTTPS. Moreover, standards such as DTLS [65] and QUIC [54] allow devices that cannot use TCP to establish similarly secure channels.

We consider three types of adversarial behavior:

- **Malicious clients:** Attempt to impersonate other clients in order to perform operations on data that is not theirs.
- **Client-side malware:** Attempts to perform unauthorized operations on client data without client's knowledge. We assume that the client's trusted device is free of malware, while all other client devices can be potentially infected.
- **Honest-but-curious servers:** Attempt to identify clients who submit requests, or to link multiple requests to the same client. Multiple servers might collude to link client requests and/or to learn client identities.

As usual, we assume all relevant cryptographic primitives are implemented and used correctly and cannot be attacked via side-channels or any other weaknesses. Similarly, we assume digital signatures can only be generated by the true owner of the private key.

Based on the above system model, we define the following requirements for VICEROY:

- **Unforgeability:** Only the client who originally interacted with the server can create a valid VCR.
- **Replay resistance:** A server will only accept a valid VCR at most once.
- **Consumer Privacy:** An honest-but-curious server (or a set thereof) should be unable to link a VCR to a specific client, or to link multiple VCRs to the same client.

IV. VICEROY Design & Challenges

This section discusses VICEROY's goals, design features, and challenges encountered. Note that we use the terms *client* and *consumer* interchangeably.

A. Design Motivation

One straight-forward way to support VCRs from accountless consumers is to require them to provide the same cookie(s) they were issued when originally visiting the server website. (Indeed this is one of the mechanisms that [64] encountered in their survey of how businesses respond to access requests.) The rationale is that only the consumer from whom the data was collected should have access to the cookie, which ties all

consumer's activity that constitutes one session. This method has several advantages: First, it is *privacy-preserving* in that, when making a request, the consumer does not reveal any further personal information that the server didn't already have. Furthermore, if the consumer submits multiple VCRs based on different cookies, the server cannot link them.[5] Second, this mechanism is easily deployable, since cookies are supported by virtually every device that uses the Web.

However, per Section I, there are also several significant disadvantages: First, this method essentially makes cookies into *symmetric* authentication tokens: anyone in possession of the cookie can create VCRs. This is problematic because cookies, in general, are not required to be unguessable and may contain predictable information, such as URLs or email addresses [52]. A subset of cookies, namely *authentication cookies*, are designed to be unguessable, but these would typically only be used once the consumer has logged into an account (i.e., no longer an accountless consumer). Second, cookies are used in all interactions with the website, and several techniques for stealing cookies have been demonstrated [19]. Third, since consumers visit many different websites, they would have to *securely* and *reliably* store a potentially large number of cookies. This differs from the usual client-side cookie management, since cookies would be additionally valuable as a means to issue VCRs. Also, if cookies are lost (e.g., due to disk failure), the consumer would be unable to exercise their GDPR/CCPA rights. This underscores the importance of cookie storage reliability. Furthermore, if the server for any reason also stores copies of cookies, same security requirements apply. If these cookies are leaked as a result of a data breach, the server would have to invalidate them in bulk, thus preventing legitimate consumers from submitting VCRs, or risk attackers requesting consumers' personal data.

B. Conceptual Design

Motivated by aforementioned challenges, we construct `VICEROY` to avoid the drawbacks discussed. We now describe key features of `VICEROY`.

Asymmetric tokens: When interacting with a server, a client provides the server with the public part of an asymmetric key-pair called the *VCR public key*. Upon receiving a VCR public key, the server associates this key with a particular *session*. Generally, a session is any linkable set of interactions between a client and a server. For example, in the Web context, a session most likely corresponds to an HTTP(S) session, which is managed using cookies. To protect the client's privacy, a new VCR public key, which is unlinkable to any previous keys, can be used for each session. Finally, to submit a VCR for a particular session, the client creates a request and signs it using the corresponding VCR private key for that session.

This approach addresses the drawbacks of using only cookies to authenticate the request, since the client's signature is assumed to be unforgeable and the client's VCR private keys are never sent over the network. It does not matter if the adversary learns the VCR public keys. The use of digital signatures also allows additional information/parameters to be cryptographically bound to the request (e.g., a request to correct personal information could, in some cases, already include the corrected information).

Cookie wrappers: At first glance, mapping data collected during a session to the VCR public key seems to be an efficient way of storing such data at server side, especially when the consumer submits a VCR. However, from a deployability perspective, it would be infeasible to replace existing Web cookies with public keys because this would require non-trivial modifications to the way servers use cookies. For example, cookie values containing URLs, email addresses, timestamps, and even JSON objects have been observed in practice [52].

To avoid changing the existing and ubiquitous cookie mechanism, `VICEROY` introduces the concept of a *cookie wrapper* – a cryptographic binding between an existing server-generated session identifier (e.g., cookie) and a client-generated VCR public key. The server generates a cookie wrapper by signing the hash of the server-generated cookie and the VCR public key using the server's long-term wrapper signing key. This allows the client to verify whether the wrapper was created correctly. This wrapper is created contemporaneously with the cookie, and at most one wrapper is created per cookie. The wrappers are then sent to and stored by the client alongside the cookie and VCR public key. The use of cookie wrappers significantly improves deployability by allowing servers to add support for `VICEROY` without modifying current cookie management.

Submitting VCRs: When the client issues a VCR signed with the relevant VCR private key (as described above), the client also sends the corresponding cookie wrapper to the server, along with the request. The server first verifies that the wrapper is valid, by verifying its own signature on the wrapper. If valid, the server then uses the VCR public key specified in the wrapper to verify the client's signature over the request. If this in turn is valid, the server is assured that this request was generated by the same client who received the original cookie (i.e., the legitimate consumer).

C. Design Challenges and Solutions

The conceptual design described above presents several design challenges. This section outlines the main challenges and presents the key insights used to realize `VICEROY`.

1) Avoiding Key Explosion: For privacy reasons, the client cannot use a static public key. A static public key would allow a server (or a set thereof) to link together multiple sessions by the same client. This linkage could take place at session initiation time, i.e., when the client requests a wrapper, or when the client issues a VCR. Also, if a client's static public key is leaked, it becomes possible to track that client's sessions globally. However, requiring each client to have a distinct public/private key-pair per session can cause a "key explosion" in which the client has to manage the large quantity of public keys and more importantly, securely store many private keys.

To avoid this issue, we use the concept of *derivable asymmetric keys*, specifically, the key derivation scheme used in Bitcoin Improvement Proposal (BIP) 32 [3]. This type of key derivation scheme allows a chain of *child public keys* to be derived from a single *parent public key*. Importantly, the derivation of public keys does not require access to

[5] Potential "fingerprinting" of the consumer's browser or network interface notwithstanding.

the corresponding parent private key. Furthermore, the corresponding child private keys can only be derived from the parent/master private key. We denote the *derivation path* of a key as `a/b/c/...`, where `a/b` is the b^{th} child key of `a`, and `a/b/c` is the c^{th} child key of `a/b`. This approach minimizes public key storage requirements of VICEROY – only the parent public key must be stored, whilst all other public keys can be derived. When a new session is initiated, the parent public key is used to generate a new child public key.

2) Multiple Devices: The client may interact with websites from multiple different devices and may subsequently want to issue VCRs for one or more of these sessions.

By design, VICEROY allows clients to use any number of devices with a single trusted device. Specifically, the master private key is used to generate a new *device public key* which is stored on each of the client's devices. The device public key can in turn be used for generating all other VCR public keys needed on the device. Note that even though the device public keys are derived from the same master private key, they are unlinkable and thus cannot be used by websites to link together sessions from the client's different devices.

3) Secure Key Management: VCR private keys for each session must be stored in a secure environment, access-controlled by the client. Leakage of private keys would allow an adversary to issue VCRs for the client's sessions, thus giving them the ability to learn, modify, or delete, potentially sensitive information.

Our use of derivable asymmetric keys reduces the number of private keys that need to be stored securely to just one – the master private key. Another benefit of using derivable asymmetric keys is that the master private key can be stored offline. This is because the master private key is only needed when generating a new device public key (i.e., when enrolling a new device) or creating VCRs, which are expected to be relatively infrequent operations.

This feature provides VICEROY significant flexibility in terms of how the master private key is stored, in order to accommodate different levels of security. For example, at one end of the spectrum, clients with low security requirements can simply store their keys on any device they trust, e.g., a phone or laptop. Clients with higher security requirements can store their keys in hardware-backed keystores, such as the Android Keystore [1] or Apple Secure Enclave [27]. On PCs, clients could make use of hardware-enforced enclaves, such as Intel SGX [17] or Windows Virtualization-base security (VBS), to protect the keys. At the top end of the spectrum, clients with the highest security requirements could store their keys in hardware security devices (e.g., YubiKey [34], Solokey [24], or Ledger [18]). These clients may also enforce additional physical security controls, such as keeping the hardware security device in a locked safe until it is needed. Clients can also make back-up copies of their master private keys to allow recovery if the trusted device fails.

We emphasize that a separate trusted device is *recommended* and not mandated. The only requirement for the trusted device is that it can derive child private keys and create signatures. Section V shows that these requirements can be met even by resource-constrained hardware security devices.

4) Long-Term Storage: A VCR may be submitted long (possibly, years) after the corresponding session ends. This typically requires clients to store state information for numerous sessions in a secure and highly available manner. Naturally, the amount of state per session must be minimal.

In VICEROY neither the cookies themselves nor the wrappers can be used to issue VCRs without a signature from the client's master private key. Therefore, the security requirements for the *storage* of cookies and wrappers are minimal – the integrity and availability of the cookies and wrappers must be maintained, but these pieces of information do not need to be kept confidential (assuming the cookies themselves have expired and are no longer useful, e.g., for authentication).

This opens up potential new business opportunities for third-party *cookie storage* providers to offer a service for safely storing cookies and wrappers on clients' behalf. This service can take care of all cookie and wrapper management as well as provide API endpoints to their customers. Note that security requirements for, and trust burden on, such services would be significantly higher if cookies alone were sufficient to issue VCRs. Also, third-party cookie storage is not a requirement; clients who are uncomfortable with third-part providers storing their cookies can store them on their local devices. Clients may also use their preferred cloud-storage service, e.g., OneDrive.

5) Broad Application Support: Many non-browser applications also communicate with application-specific servers, which, similarly to Web servers, collect consumer data. VICEROY is sufficiently flexible to be used in these applications as well. This is achieved through the design pattern of using wrappers instead of directly modifying the session identifiers. For example, if a non-browser application uses a different form of session identifier (i.e., not a Web cookie), VICEROY can still be used since the wrapper can be used to cryptographically bind the client's VCR public key to any type of session identifier.

D. Overall VICEROY Design

Bringing together the design concepts discussed in the preceding sections, this section presents the overall design of VICEROY. The precise protocol messages exchanged between the various principals are shown in Figure 1 and the various cryptographic keys are defined in Table I.

1) Setup and device provisioning: The client first generates a master private key $sk(t)$ on a trusted consumer device t. This key is then used to derive a device-specific public key for each of device the client will use for interacting with websites, e.g., $pk(t/i)$ for device i. Device public keys are provisioned using a simple request-response protocol, as shown in Figure 1 (**A**). The device public keys are stored within the respective devices for rapid future VCR key generation.

2) Website Interaction: When initiating a session with a server, upon a client request, the server generates a unique client id and sends it to the client in the form of a cookie. The client generates a new VCR public key $pk(t/i/j)$ for session j, derived from the device public key. The client then sends this newly-generated key, along with the client id cookie, to the server as shown in Figure 1 (**B**). After receiving the VCR key, the server signs the hash of the VCR key and cookie

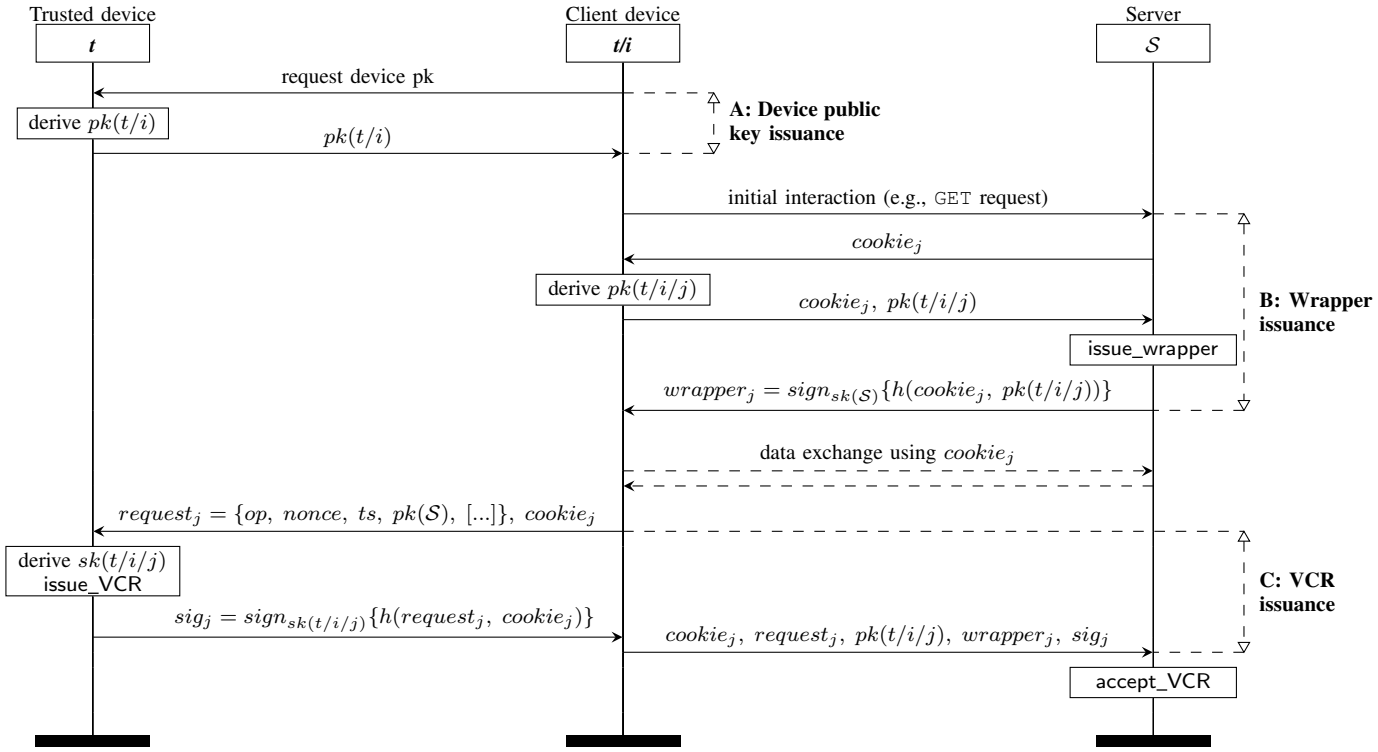

Fig. 1. Protocol messages exchanged in VICEROY. $sign_k(m)$ denotes a cryptographic signature on message m using key k, and $h(m)$ denotes a cryptographic hash of message m. The events issue_wrapper(j), issue_VCR(j), and accept_VCR(j) are used in the formal security analysis in Section VI.

TABLE I. LIST OF KEYS AND THEIR ROLE IN VICEROY

Key	Name and description
$sk(t)$	Master private key: generated and stored in trusted device.
$pk(t/i)$	Device i public key: derived from master private key $sk(t)$ within the trusted device, and stored on device i.
$pk(t/i/j)$	VCR j public key: derived from device public key $pk(t/i)$ when requesting a new wrapper.
$sk(t/i/j)$	VCR j private key: derived from master private key $sk(t)$ within the trusted device when issuing a VCR.
$sk(\mathcal{S})$	Wrapper signing key: long-term private key used by server when generating a wrapper.
$pk(\mathcal{S})$	Wrapper public key: long-term public key used by client when verifying a wrapper. Obtained via standard PKI.

to generate a wrapper. Then it returns the wrapper to the client, attesting to the association between the client id cookie and the VCR key. To check the integrity of the wrapper, the client verifies the signature using the server wrapper public key which can be obtained and verified the same way it obtains and verifies the server's TLS public key.

3) VCR Issuance: As shown in Figure 1 (**C**), the client generates a request, which consists of the specified operation (e.g., retrieve, modify, or delete), the current time, and any optional parameters. The client uses the master private key $sk(t)$ on the trusted device to derive the respective signing key $sk(t/i/j)$ for device i and session j. The signing key $sk(t/i/j)$ is then used to sign a hash of the request and the associated cookie, which is returned to the client. The client then sends this signature, along with the cookie, request, public key, and wrapper to the server, constituting the VCR.

Upon receiving a VCR, the server first verifies its own signature on the wrapper to confirm the authenticity of the wrapper. The server then verifies the client's signature on the VCR using the public key $pk(t/i/j)$ from the wrapper. If these checks succeed, the server accepts the VCR and proceeds with the requested data operation.

To prevent an attacker replaying a valid VCR to the server, the client includes a unique $nonce$ in the request, which is thus also included in the client's signature sig_j in Figure 1. Upon receiving this VCR, the server checks that it has not already processed a VCR containing that $nonce$. The client also includes a timestamp ts in the request, representing the time at which the VCR was issued. Each server defines its own recency threshold (e.g., 12 hours) and rejects any VCRs that are older than this threshold. This means that the server only has to store nonces for up to this threshold in order to check that new requests are unique. This also ensures that an attacker cannot delay valid VCRs arbitrarily (e.g., if the attacker were able to block a VCR and then release it months later, this could have unintended consequences for consumers).

An alternative would be to use a challenge-response protocol where the server generates a challenge that the client must include in the signed VCR. This would avoid the need for a nonce and timestamp, but would increase the complexity of the system and could be abused to mount a denial of service attack against the server, similarly to a TCP SYN flooding attack (although well-known cookie-based countermeasures

and application level solutions such as CAPTCHAs [70] or rate-limiting mechanisms [61] could also be used.).

4) Device Unprovisioning: Finally, the full device lifecycle may require *unprovisioning*, e.g., if the device is lost/stolen, or sold/recycled. We separately consider the implications for a regular or trusted device.

Regular Device: Since such devices do not hold private VCR keys, unprovisioning is straight-forward. The client only has to unlink the old device from the respective trusted device to prevent any further VCR issuance. This can be done from the trusted device, even if the old device is lost/stolen. The client may wish to back-up or transfer any cookies and wrappers from the old device.

Trusted Device: From an availability perspective, the client should be able to recover the master private key from a backup. From a security perspective, the trusted device should ideally have some type of access control (e.g., using a PIN and/or a fingerprint) to protect the private key even from an adversary who has physical access to the device. If the trusted device is being sold/recycled, the client should securely back-up or transfer the master private key to a new trusted device using techniques such as Presence Attestation [72].

V. Implementation

We now describe our implementation of `VICEROY`, which consists of: a browser extension, a trusted device implementation, and a modified web server.

A. Browser Extension

This component provides most of the client-side functionality on a regular consumer device. Specifically, it manages VCR public keys, implements client-side aspects of the VCR flow, and includes a client-facing interface for controlling this flow. It also handles communication between the browser and the trusted device (or application) that holds the master private key. Although we chose to implement a proof-of-concept extension for Google Chrome, no (or only minor) mods would be required to get it to work with any modern browser.

To realize derivable asymmetric keys, we used a mechanism proposed for hierarchical deterministic wallets, commonly known as Bitcoin Improvement Proposal 32 [3], or `BIP32`. `BIP32` has the notion of an *extended key*, with 256 bits of entropy (called *chain code*) added to a normal public/private key-pair. Extended keys can be used to derive one or more child keys, following the rule that private keys can be used to derive private or public keys, while public keys can only derive public keys.[6] `BIP32` is also well-suited for low-end devices, since it was designed for (resource-constrained) Bitcoin hardware wallets. Section V-C describes our proof-of-concept of a resource-constrained trusted device.

The browser extension comprises a background and a pop-up scripts, both written in `JavaScript`, with additional `HTML` and `CSS` for the pop-up. As no `JavaScript` version of several `Node.js` libraries (e.g., `crypto`, `BIP32`) was available, we used Browserify [8] to convert such libraries into `JavaScript` files that can be loaded by the browser.

1) Background Script: This component houses most VCR-side functionality. It uses the browser's API (`chrome.webRequest.onHeadersReceived`) to scan `HTTP` response headers to detect which servers support `VICEROY`. If present, it parses the relevant `VICEROY` endpoints and the client id cookie from the headers.

Using the device public key, it derives a session-specific VCR public key using a port of the official `BIP32` implementation [4]. The derivation path of a VCR public key is in the form `t/i/j`, where `t` denotes the master private key, `i` the device ID, and `j` the total number of sessions created on the device. In other words, `t/i` represents the device public key and `t/i/j` represents the child public key for the j^{th} session.

After deriving a VCR public key, the background script either includes this in the next request header sent to the server, or sends a `POST` request to a separate server-defined VCR wrapper request endpoint (we implemented the latter). The server then responds with a newly generated wrapper.

The background script then stores the server-returned wrapper along with the key derivation path. Since we generate a new one for each session, the number of stored wrappers may grow large, depending on how many new sessions the client establishes. However, the storage overhead of the wrappers is not significant, as the number of wrappers is at most the same as the number of cookies the client must store (see Section VI-D for client-side storage evaluation). To improve efficiency of searching for a client id cookie, we use a hashmap of client id cookies and their corresponding wrapper information. We also store the URL of the website and the time of the visit alongside the wrapper to assist clients when selecting a session during VCR issuance. The background script can store this data using any storage service. Our implementation uses the local storage API (`chrome.storage.local`). Other choices include cloud storage services or Google Chrome's synced storage API (`chrome.storage.sync`), which would allow clients to synchronize `VICEROY` data between different devices.[7]

Note that all the above operations are performed asynchronously, in the background. Thus, the client does not experience any additional latency in loading the page.

2) Pop-up: The extension pop-up is a small web page that appears when the client clicks on the extension icon next to the URL bar. As shown in Figures 2 and 3, it displays session information for a VCR key along with the history of web links visited when the session was active. It also displays the types of VCRs (access, modify, and delete) that the client can submit.

To issue a VCR, the client first chooses the session(s) and the type of request. Next, the pop-up script prepares a request for signing. It includes a timestamp in the signature to prevent replay attacks, due to its simple design. The resulting client request is then passed to a Python native application [21] which relays it to the trusted device (see Section V-C) for signing. Once signed using the VCR private key, the request is retrieved by the pop-up (using the background script) through

[6] `BIP32` also has the notion of *hardened* vs. *non-hardened* keys, though we only use the latter. The difference between these two is the algorithm used when deriving them from the parent key.

[7] Unfortunately, we found that Chrome synced storage currently imposes a limit on the amount of stored data.

Fig. 2. `VICEROY` browser extension pop-up displaying multiple sessions. `SID` is the first few bytes of VCR public key.

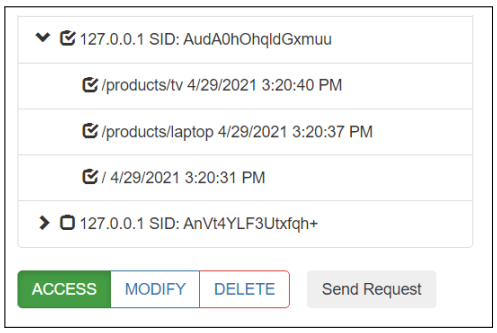

Fig. 3. `VICEROY` browser extension pop-up displaying the history of visits in each session. Clients can select which session and which type of VCR (`ACCESS/MODIFY/DELETE`) they wish to generate.

the native application, and the VCR is sent to the server's VCR verification endpoint. Finally, the server's response is displayed in the client's browser.

B. Native Messaging Application

To simulate a client-controlled trusted device, we created a `Node.js` [22] application that holds the master private key. Alike the trusted device, this application signs VCRs received from the background script using private keys derived from the master private key with the provided key derivation path, e.g., `t/0/1`. Communication between this application and the browser is done via the native messaging protocol [21]. We use the `native-messaging` package [20] to support both Firefox and Chrome. As mentioned in Section IV-C3, the key storage mechanism must support derivable asymmetric key operations (e.g., using `BIP32`). There are publicly available implementations of `BIP32` in different languages, including: JavaScript [4], Golang [5], Python [7], Java [6] and C [13].

C. Trusted Device

We implemented a proof-of-concept trusted device using the Solokey Hacker [24], a security token implemented using open source code [25] and hardware [26]. Solokey includes an STM32L432KC microprocessor with an Arm Cortex-M4

Fig. 4. Solokey hardware security token in use.

Fig. 5. Solokey token compared to a standard AA battery.

MCU (80MHz), 64 kB of RAM, 256 kB of flash memory, a true random number generator (TRNG), and a physical button for presence attestation, as shown in Figures 4 and 5. We extended the FIDO2 Client to Authenticator Protocol (CTAP) API on the device with the following three calls:

- `KEYGEN`: Generates a master private key using the TRNG. This key never leaves Solokey.
- `DEVKEY`: Takes a key derivation path (`t/i`) and outputs a generated device public key (see Figure 1 (**A**)).
- `VCRGEN`: Takes a key derivation path (`t/i/j`) and a consumer request (in the form of a cryptographic hash), and outputs a signed VCR (see Figure 1 (**C**)).

All three trusted device API calls require human confirmation achieved by the client pressing the physical button on the Solokey. For key derivation and signing we used the `BIP32` implementation from Trezor [32] due to its popularity and suitability for embedded devices. The native messaging application calls the individual API according to the command it receives from the background script. Once the client confirms the action by pressing the button and the native messaging application receives a response from Solokey, the application relays the response back to the background script.

To protect against tampering with the consumer request by client-side malware, the trusted device could also display information about the VCR that is about to be generated, including the request type, the website URL, and the timestamp. In our prototype, this could be as simple as changing LED color on the Solokey. Displaying more detailed information about the request may require trusted devices to have a dedicated, human-perceivable output means, e.g., a display.

D. `VICEROY`-enabled Web Server

We implemented a proof-of-concept `VICEROY`-enabled `HTTP` server using the Express Node.js web framework [10]. It uses `HTTP` sessions and indexes data collected during each session using session cookies. For signing, it uses ECDSA with curve `secp256k1` [30], although any secure signature scheme can be used.

When the client first visits a web page hosted on our server, the latter creates an `HTTP` cookie that includes a client id (uuid [23]).Hereafter, all data collected by the server about this

client is associated with the client's unique id.[8] In response to the initial client request, the server notifies the client that it supports VICEROY.

The server provides the client with an HTTP endpoint for obtaining a wrapper. The client's browser extension sends the client id cookie set by the server and a freshly-generated VCR public key to this endpoint, and the server then generates a wrapper that cryptographically binds these two pieces of information. This endpoint can be configured to only issue wrappers for a short duration after the cookie was issued, to avoid adversaries attempting to bind their own public keys to arbitrary cookies. Alternatively, the client can provide a freshly-generated VCR public key in the next HTTP request, and the server can issue the wrapper in the next response.

The server also provides the client with a VCR endpoint, to which the client can later submit VCRs for this website. As discussed in Section IV-D, the client sends a wrapper and a signed VCR, which the server then verifies. To support requested VCR actions, the consumer's request may include metadata specific to the requested action. For instance, for data access requests, metadata may include an encryption key to be used by the server to encrypt data to be returned to the client.

VI. Evaluation

We now evaluate security of VICEROY as well as its latency, data transfer, and storage requirements.

A. Security Analysis

To evaluate security of VICEROY, we defined a formal specification of the protocol using the Tamarin prover [60], [28]. The full specification is provided in Appendix X. In Tamarin, a protocol P is modelled as a set of labelled transition rules operating on *facts*. A transition consumes linear facts from state s_{i-1}, generates new facts for state s_i, and labels the transition as a. An execution of protocol P is a finite sequence of states and transition labels $(s_0, a_1, s_1, \ldots, a_n, s_n)$ such that $s_0 = \emptyset$ and $s_{i-1} \xrightarrow{a_i} s_i$ for $1 \leq i \leq n$. The sequence of transition labels $(a_1, \ldots, a_n)$ is a *trace* of P and the set of all traces is denoted $traces(P)$. Security properties are specified using first-order logic formulas on traces.

Unforgeability: Using this specification, we formally define the security property of *unforgeability* (see Section III), for both wrappers and VCRs. As shown in Figure 1, let $\mathsf{issue\_wrapper}(s, o, pk)$ denote server s issuing a wrapper for cookie o and public key pk (corresponding to private key sk); let $\mathsf{issue\_VCR}(c, o, pk, r)$ denote client c issuing a VCR for request r, cookie o, and public key pk; and let $\mathsf{accept\_VCR}(s, o, pk, r)$ denote server s accepting a VCR for request r, cookie o, and public key pk.

Definition 1 (Wrapper unforgeability)**.** *A protocol P satisfies the property of* wrapper unforgeability *if for every $\alpha \in traces(P)$:*

$$\forall s, o, pk, r, j.\ \mathsf{accept\_VCR}(s, o, pk, r) \in \alpha_j \implies \exists i.\ \mathsf{issue\_wrapper}(s, o, pk) \in \alpha_i \wedge i < j$$

[8] VICEROY can also use any existing client identifier cookie scheme – such as Google Analytics' _ga and _gid cookies [12].

Definition 2 (VCR unforgeability)**.** *A protocol P satisfies the property of* VCR unforgeability *if for every $\alpha \in traces(P)$:*

$$\forall s, c, o, pk, r, j.\ \mathsf{accept\_VCR}(s, o, pk, r) \in \alpha_j \implies \exists i.\ \mathsf{issue\_VCR}(c, o, pk, r) \in \alpha_i \wedge i < j$$

The Tamarin prover verifies that, in the protocol as specified, both of these properties hold for an unbounded number of protocol runs (the strongest possible result). Since wrappers and VCRs are ultimately verified by the server, Definition 1 requires that, whenever a server accepts a VCR, that server must have issued a corresponding wrapper at some prior time point. Similarly, Definition 2 requires that, whenever a server accepts a VCR, there must exist a client that issued that VCR at some prior time point. Intuitively, these properties show that the adversary cannot forge a valid wrapper or VCR for a given pk. Since the corresponding private key is only known to the client, we conclude that the unforgeability property is satisfied.

Replay resistance: To prevent an attacker obtaining a genuine VCR (e.g., by eavesdropping) and later replaying it to the server, we formally define an the security property of *replay resistance* (see Section III) for VCRs.

Definition 3 (Replay resistance)**.** *A protocol P satisfies the property of* replay resistance *if for every $\alpha \in traces(P)$:*

$$\forall s, o, pk, r, j.\ \mathsf{accept\_VCR}(s, o, pk, r) \in \alpha_i \wedge \mathsf{accept\_VCR}(s, o, pk, r) \in \alpha_j \implies i = j$$

The Tamarin prover verifies that this property holds for an unbounded number of protocol runs. Definition 3 states that, if there are two trace events in which a server accepts the same VCR, these must be the same event. Since Tamarin does not model time-based properties, the formal model uses only a nonce in the VCR to check for uniqueness. In practice, the client would also include a timestamp in the VCR, as described in Section IV-D3.

Consumer/Device/Request Linking: To protect clients' privacy, an honest-but-curious server should be unable to link a VCR to a specific client, or to link multiple VCRs to the same client (see Section III). This requirement ensures that the use of VICEROY does not reveal any additional information to the server about potential links between users, devices, and sessions (e.g., if a single user is using multiple devices).

Since unlinkability is not a trace property, we cannot use Tamarin to model or verify this property. Instead we follow an existing approach for reasoning about unlinkability [69], [68], [63] and show that the messages sent by the client to the server do not contain any information that could be used by a server to link VCRs to clients or to other VCRs.

As shown in Figure 1, the only new pieces of information provided by the client (which the server does not already know) are (1) the $request$, (2) the VCR public key ($pk(t/i/j)$), and (3) the client's signature. The $request$ does not contain any information that uniquely identifies the client or allows it to be linked to other requests. Formally, given any two requests, a server would not be able to distinguish whether or not they were issued by the same client. BIP32 guarantees that derived public keys are unlinkable to each other and to their parent keys. This ensures that neither the VCR public key

nor the signature created using the corresponding private key are linkable.[9] Formally, given any two derived public keys or signatures, a server would not be able to distinguish whether or not they were issued by the same client. We can therefore conclude that the protocol satisfies the *unlinkability* property.

Of course, if the device public key is leaked from the client's device, different VCR could be linkable. However, even in this case, VICEROY is no worse than the current use of web cookies, since an attacker that can steal a device public key could also steal cookies from the victim's device and use these to link/track the victim's sessions and VCRs.

Although VICEROY provides unlinkability by design, the nature of how VCRs are submitted may point servers in the direction of clients. For instance, by observing metadata such as IP addresses of different VCR requests, a server might link them to the same client. One possible mitigation is to use anonymity networks (e.g., Tor [31]) and avoid issuing VCR *bouquets* whereby multiple requests are submitted through the same connection. Random delays between VCRs can be used to prevent timing-based correlation.

Public Key Injection: The adversary might attempt to replace the client VCR public key with its own public key when obtaining a wrapper for a cookie. This can occur if there is either: *(1)* an active network-level adversary, and/or *(2)* malware on client device. For (1), this is mitigated by the use of secure communication channels (e.g. TLS) or by simply having VICEROY browser extension compare the public key it sent with the public key in the returned wrapper. In contrast, (2) is difficult to defend against. A malware in full control of the client device, can replace the client public key with its own during the wrapper request. Even if the consumer attempts to verify the public key in the returned wrapper, the malware can subvert this check. The only way to prevent this is to verify wrappers in an environment isolated from client-side malware, e.g., a Trusted Execution Environment (TEE). Of course this approach would also require securely sharing the public key to be verified with the TEE and displaying the verification result to the user without malware interference. Overall, we consider (2) to be out of scope since malware in total control of a client device already has access to any data that could be collected by the server about the client.

Client-Side Malware: Malware on the client's device might conduct unauthorized operations on client data. This might be possible either via: *(1)* replay attacks, or *(2)* by generating VCRs without the owner's consent. This issue highlights one important difference between using asymmetric tokens and symmetric tokens for VCRs. The former allows generation of one VCR per client authorization. In contrast, symmetric tokens, even if encrypted and decrypted on demand with client's approval, need to be available in plaintext at some point in order to be sent to the servers. Client-side malware can use these exposed tokens to generate future VCRs. Also, using symmetric tokens allows malware to access data from *before* its infection period. For example, assume that malware infects the client's device at time t. It can access pre-stored tokens and learn data generated *prior* to t. Since we can prevent such attacks via asymmetric tokens, VICEROY provides better security and privacy compared to current symmetric token-based systems.

[9]By design, a VCR can be linked to the corresponding wrapper– indeed the latter is included in the former.

TABLE II. LATENCY RESULTS FOR VICEROY WRAPPERS.

Key Derivation	Wrapper Generation	Wrapper Verification	Wrapper Storage
24.6 ms	0.4 ms	18.8 ms	6.5 ms

Key Leakage: An attacker might exploit weaknesses in BIP32 to learn the private key. One well-known weakness of BIP32 is that knowledge of a parent extended public key as well as of any non-hardened child private key (descended from that parent public key) can leak the parent extended private key [71]. However, in VICEROY, the non-hardened child private key is generated within, and never leaves, the trusted device. Therefore, the attacker must compromise the trusted device to obtain the non-hardened child private key, which we consider to be infeasible, per Section III.

B. Latency Analysis

Most operations in VICEROY result in no user-perceptible latency because they occur asynchronously with normal web browsing. Nevertheless, we discuss them to quantify computational costs of VICEROY. The only user-perceivable latency occurs when a VCR is issued, which is expected to be an infrequent operation. For these experiments, we used as the client device an Intel NUC with an Intel Core i5-7260U 2.20GHz quad-core CPU with 32.0 GB of RAM running Ubuntu 18.04 LTS, with Chrome version 97.0.4692.71 64-bit official build. Unless otherwise stated, all results are averages over 10 runs, with storage left unchanged between runs. All data was in local storage and results may vary depending on the underlying storage technology, e.g., memory vs. hard-drives vs. cloud-hosted databases.

Obtaining a Wrapper: We divide the process of obtaining a wrapper into the following four phases:

1) *Key Derivation*: When an unknown client makes a request, the server returns a cookie and the VCR endpoints. The client parses these endpoints, derives a VCR public key, and prepares a wrapper request.
2) *Wrapper Generation*: The server generates a wrapper using the client-provided VCR public key and cookie.
3) *Wrapper Verification*: After receiving the wrapper from the server, the client verifies the wrapper using the server's public key and confirms that the wrapper associates the correct VCR public key and cookie.
4) *Wrapper Storage*: The client saves the wrapper along with the public key derivation path and endpoints.

Table II shows the average time of each phase. These measurements exclude network latency, as this will vary depending on the locations of the client and server. Wrapper Verification takes the longest as it includes a signature verification. Wrapper Generation is noticeably faster than Key Derivation and Wrapper Verification since the former is performed by a native application and the latter two run in the browser extension.

Issuing a VCR: We divide the process of issuing a VCR into two steps:

TABLE III. VCR LATENCY RESULTS.

	VCR Generation	VCR Verification
VCR Flow	1357.4 ms	1.5 ms

TABLE IV. DATA TRANSFER IN KB (HTTP HEADER + PAYLOAD).

	Request	Response	Total
Obtain Wrapper	0.72 kB (0.62 + 0.10)	0.38 kB (0.23 + 0.15)	1.10 kB
Issue VCR	0.99 kB (0.68 + 0.31)	0.28 kB (0.23 + 0.05)	1.27 kB

VCR Generation: When a client selects a session and a VCR type, the browser extension prepares a request to be signed by the trusted device and a key derivation path. Both are sent to the trusted device via the native messaging application. Next, the trusted device signs the overall request using the private key corresponding to the derived public key. Finally, the signature is then returned to the extension. The above steps in total take on average 1357.4 ms using a modified Solokey Hacker as the trusted device.

VCR Verification: The server receives the VCR and verifies the wrapper. Also, it extracts the VCR public key for this session from the wrapper and verifies the overall VCR using the VCR public key. This step takes on average 1.5 ms.

Table III shows latency results (excluding network latency). Similar to Table II, a signature generation operation performed by the native messaging application takes longer, compared to a standalone server. This is due to data passing delay between pop-up and background scripts, as well as the native messaging protocol between the application and the browser. Based on these results, server's cost to verify VCRs is minimal.

Trusted Device Latency: Finally, we benchmarked the latency of key-generation operations performed by the trusted device. Generation of the master private key takes 332 ms and generation of a device key takes 724 ms, which are both reasonable for these types of operations. As discussed in Section V, we use a modified Solokey Hacker as an example of a trusted device, while noting that this resource-constrained hardware token is likely to be the slowest type of a trusted device. We emphasize that these are very infrequent operations, taking place once per trusted device, and once per new client device respectively.

C. Data Transfer Analysis

We measured the amount of data transferred to obtain wrappers and issue VCRs. For demonstration purposes, our server kept the visit history for each client, which was returned to the client upon successful VCR verification. The setting was the same as for latency analysis, and we used the browser's debugging console to measure the amount of data exchanged between the browser and server.

The client first sent an `HTTP GET` request to the server. After receiving the VCR endpoints and a cookie, the client sent a `POST` request to the wrapper request endpoint. The client then generated and sent a VCR to the server requesting to access the data, and received the visit history with a single entry. This request included: wrapper, VCR public key, and signature on the request. Table IV shows the `HTTP` header and payload sizes transmitted between the client and server. For both obtaining a wrapper and issuing a VCR, the amount of data exchanged was a fraction of a typical web interaction.

D. Storage Analysis

For the client-side data storage evaluation, we measured client-side data storage requirements using the `chrome.storage.sync.getBytesUsed` function. The date was represented using the UNIX standard and the only the URL path was stored in the history section.

We first measured the minimum storage for a client with no visit history: it stores server endpoints, VCR and server public keys, plus metadata used to issue VCRs, requiring 0.38 kB in total. We then measured storage for a client who visited a particular URL 100 times, where `VICEROY` stored the details of each visit. This required 5.06 kB. Although storage increases linearly with the number of visited more web pages, the gradient is small and the overall magnitude is similar to that of a typical web browser history.

We can extrapolate from the results above to estimate the amount of storage required by a typical client to store all wrappers generated over a long period of time. Crichton et al. [45] recently found that, on average, a user visits 163 distinct web pages per day. Note that "distinct web page" refers to a unique URL and not necessarily to a unique domain, i.e., the number of distinct domains may be smaller. This study does not report the fraction of web pages where a user has an account. We make the following conservative assumptions: (i) the 163 web pages correspond to distinct domains; (2) the user has no accounts on any of these web pages; and (3) the user stores all wrappers for one year before issuing a VCR request.[10] Under these assumptions, `VICEROY` would require 0.38 kB for each of the 163 web pages for 365 days, resulting in a total storage requirement of 22.61 MB.

E. Deployability Analysis

From the server perspective, main changes are: (1) create and maintain a public/private key-pair for generating wrappers, and (2) create and maintain relevant endpoints. By design, the server does not have to change how it assigns identifiers to clients or uses cookies. The integration of `VICEROY` into existing servers can be further simplified by releasing `VICEROY` modules for popular server frameworks. From the server's perspective, `VICEROY` is a cheaper and simpler approach to comply with data protection regulations, compared to existing third-party identity verification services.

From the client's perspective, `VICEROY` requires: (1) generate a master private key on a trusted device, and (2) install the browser extension and native messaging application on other devices. These software packages could be made available via the popular app stores. Once installed, `VICEROY` operates transparently to the client, and does not disrupt the normal flow of web browsing. The anticipated incentives for clients to use `VICEROY` are that it is both more automated and more privacy-preserving than current identity verification methods.

[10] Note that this analysis is purely illustrative and that the storage requirements could increase or decrease if these assumptions change.

VII. Discussion

A. Multi-Device Support

Given the proliferation of smartphones, computers, and various IoT gadgets into many spheres of everyday life, we expect that most clients own (or soon will own) multiple devices with varying capabilities. VICEROY has been designed with this scenario in mind. First, computational requirements of VICEROY can be met by any device that can establish TLS connections. Any device that can perform a TLS handshake is sufficiently powerful to verify the signature on a wrapper. Second, storage is not an issue because wrappers can be stored anywhere. This also allows devices without a display to request wrappers and commit these to synced storage. Thereafter, any other device with an appropriate display owned by the same client can fetch these and perform data operations. Third, the use of BIP32 allows VICEROY to generate an arbitrary number of device public keys, which can be used to derive any number of VCR public keys for interaction with different websites. Since private keys are not stored on such devices, there is no threat against security of VICEROY, even if the number of devices increases. Fourth, VCRs do not need to be issued from the same device that originally interacted with the server. Devices may need to be unprovisioned, but the client should still be able to issue VCRs for interactions they made on those devices. In VICEROY, the client can always transfer cookies and wrappers between devices.

B. Multi-VCR Support

Unlinkability of VCRs is critical for protecting consumers' privacy (as defined in Section III). However, it requires clients to generate and sign VCRs for each session. To reduce overhead, a client can amend its key derivation mechanism. For example, if a client prefers to use only one VCR to refer to combined collected data for all sessions with a particular website, it can use the derivation path `t/i/s/j` where `t/i` is the derivation path of the device key as before, `s` is a server id and `j` is a server-specific (rather than global) session counter. The client then collects all wrappers and generates a unified VCR by signing it with the private key corresponding to the server VCR public key (`t/i/s`). This server VCR public key is then sent to the server. The server derives VCR public keys for individual sessions and verifies all wrappers. For a new session with the same server, the client simply updates the server id (`s`) and repeats the process.

C. Multi-Communication Protocol Support

So far, we focused on VICEROY being used over HTTP(S), the most common way to access Web services. However, it can support any stateless protocol that assigns a unique identifier by using that identifier when generating a wrapper. Thus, as described in Section IV-C5, VICEROY is also applicable to applications that use other protocols to interact with online servers. Such applications can use VICEROY wrappers to bind client-generated public keys to any type of symmetric session identifier, and the same protocol to issue VCRs for data associated with that identifier.

D. Shared Devices

Certain types of client devices may be shared by multiple individuals, e.g., a smart TV used by all household members. In the worst case, it may not be possible to associate usage data with a specific individual, e.g., if two or more people are using the device concurrently. This a general *policy* question in the field of data protection; it is not unique to VICEROY. There is no clear guidance in either GDPR or CCPA as to how to handle such situations. However, VICEROY provides some mechanisms that could be used to assist with *enforcing*, rather than *defining*, data ownership policies for shared devices. For example, one conceivable data ownership policy for a shared device is that *all* users of the device must consent to VCRs being issued for sessions originating from this device. In the example above, this would require all smart TV users to consent to a data access request, which may also help address the policy question of who actually *owns* the data. This could be achieved by using a signature scheme which requires all users participating in the creation of a VCR. The actual policy and procedures regarding data from shared devices should be defined by the regulators, while tools such as VICEROY should enable, rather than dictate, policy.

E. 3^{rd} Party Storage

One distinctive feature of VICEROY, as opposed to simply using cookies, is that possession of a wrapper alone is insufficient to issue a VCR. Thus, wrappers can be stored by third parties and retrieved only when needed. This relaxes client-side storage requirements and creates a possible new business opportunity for (paid) service providers that manage wrappers on behalf of clients.

F. Broad Identifier Support

VICEROY wrappers are a general means of binding client identification cookies to client-generated public keys. Importantly, this method is *non-invasive* and does not impose any constraints on the cookie, which is useful if a server changes its identification method, e.g., cookie content. VICEROY can also support future identification methods, as a server can simply issue wrappers that bind public keys to any new type of identifier, instead of the cookies.

G. 3^{rd}-party Cookie Support

Third-party cookies are claimed to provide better, more personalized advertisements. Although such cookies are commonly considered detrimental to privacy [59], they are widely used; around 79% of 109 million web pages include third-party cookies [41]. VICEROY can support such cookies by modifying the browser extension to capture traffic going to third parties and extract and store all third-party cookies. To obtain the wrappers, the client can either: (1) visit the third-party wrapper endpoints individually, or (2) send all cookies to the first-party server, which would obtain wrappers on the client's behalf.

A related aspect is the technique commonly known as *cookie syncing* [35], [43], [9], [67]. Instead of placing multiple cookies on the client device, a set of servers associate the data they collect under a unified identifier. VICEROY can support this type of cookie by having the client visit wrapper endpoints

of all involved third-parties, or by having the first-party server visit them on the client's behalf.

Note that VICEROY does not require enabling third-party cookies in order to function. This is related to *cookie syncing* [35], [43], [9], [67] in which, instead of placing multiple cookies on the client device, a set of servers associate the data they collect under a unified identifier. VICEROY can likewise support this type of cookie and obtain the wrapper by visiting the relevant endpoint.

H. Further privacy considerations

When a consumer issues a VCR for a particular session, a potential risk arises that this action could reduce consumer privacy by allowing the service provider to link multiple sessions to the same consumer. For example, there is a one-to-one mapping of a VCR public key and a session, thus revealing VCR keys to the servers might allow servers to link VCRs as well. To prevent this, we consider two approaches:

First, for data access requests, cryptographic techniques, such as Private Information Retrieval (PIR) [44], can hide the identity (i.e., VCR public key) of the data requested. For modify and delete operations, the problem is more challenging, because, if data is in plaintext, servers could detect what has been updated/deleted. Furthermore, PIR is likely to incur a high bandwidth burden, since databases may retain data for very long periods of time (e.g., 10 years) and that large database might need to be sent to the client.

An exciting approach to overcome both these problems is to introduce Trusted Execution Environments (TEEs) on the server side. Using remote attestation [16], after ensuring that expected code is running on a server-side TEE, clients can create a secure channel to the TEE and send their VCR keys over it. The TEE can find the matching row in the database and return associated data. Recent versions of Intel Software Guard Extensions (SGX) [33] support up to one terabyte of enclave memory, where this database can be stored. This TEE-secured database can be populated with data collected during a secure connection between a client and a server, e.g., using techniques such as LibSEAL [37]. Furthermore, Intel SGX DCAP [15] supports attestation reports that attest the origin of the report (i.e., the server) rather than only attesting that the TEE is *a genuine Intel SGX device*.

Server-side TEEs also allow servers to prove to clients how their data is used, also using remote attestation. VCR responses can be generated with such guarantees, providing more transparency and trust between clients and servers.

I. Further Applications

Although VICEROY focuses on VCRs from accountless clients, it can also supplement verification of VCRs from account-holding clients. This can be useful, considering that passwords suffer from dictionary attacks and are often re-used on multiple servers.

VICEROY can also be used as a basis in scenarios that require client re-authentication. For example, in the context of monetary transactions, receipts are currently used to prove that a client bought something from a merchant (server) in order to accept returns or perform exchanges. With VICEROY, a client can supply a fresh VCR public key during the purchase transaction and later generate a proof of ownership of the corresponding private key, which would anonymously confirm to the server that this is indeed the same customer.

VIII. Related Work

Supporting VCR requests from accountless consumers. Until now, the only means of authenticating accountless consumers have been *ad hoc*. [64] reports that such means may require one or more of: device cookies, government-issued IDs, signed and witnessed statements, utility bills, credit card numbers, or participation in a phone interview. However, these mechanisms are burdensome for consumers. Furthermore, they are insecure (as shown in [64], [40], [47], [39]), error prone (due to the manual processing) and privacy-invasive due to the additional information collected. In contrast, VICEROY allows consumers to submit VCRs in a secure and private manner without requiring any human interaction on the server side.

Security of GDPR Subject Access Requests. As described in Section II, the GDPR and CCPA grant subjects the rights to request access to their personal data collected by businesses, by submitting a VCR or Subject Access Request (SAR). Unfortunately, insecure (or easily circumventable) SAR verification practices open the door to potential leakage of personal data to unauthorized third parties. Prior work [40], [47], [64] has investigated various social engineering techniques for bypassing existing SAR verification practices.

For example, Cagnazzo et al. [40] demonstrated that an unauthorized adversary can abuse the functionality provided by a business to update a victim subject's address (for both email and residential addresses). The adversary could then request access to *"their"* data from this new address. Out of 14 organizations tested, 10 gave out personal information and 7 of these contained sensitive data.

Di Martino et al. [47] investigated the use of address spoofing techniques (e.g., using homoglyphs), as well as more sophisticated techniques such as manipulation of identity card images. It was found that 15 out of 41 organizations with manual verification processes leaked personal data. The remaining 14 organizations required an account-based login, which was impervious to such attacks, but is not available for the accountless consumers we consider in this work.

Pavur and Knerr. [64] performed an extensive evaluation of 150 companies' practices for SAR verification. Results indicated that email address-based and account login were the most common, followed by device cookies, government IDs, and signed statements. Some organizations also requested utility bills, phone interviews, or credit card numbers. To bypass SAR verification, [64] created and sent a vague SAR letter to organizations. Out of 150, 24% disclosed personally-identifying information.

Boniface et al. [39] also analyzed SAR verification practices for popular websites and third-party trackers. The findings were that, in addition to possibly being insecure, SAR verification could undermine the privacy of subjects in order to verify the request.

General Studies on GDPR Subject Access Requests. Urban et al. [66] performed a two-sided study of both data

subjects and data-collecting organizations, with a focus on online advertising. For data subjects, consumer surveys were used to evaluate the usability of data transparency tools offered by the organizations, and to learn more about consumers' perceptions of these tools. [66] also conducted surveys and interviews of organizations to get their views on the privacy regulations and business practices for SARs. The results paint a picture of discrepancy between the consumer' perspectives and the collected data, which is also corroborated by Ausloos and Dewitte [38]. Furthermore, consumers seemed to show little interest in seeing raw technical data. Similarly, Urban et al. [67] investigated SAR practices for online advertising companies and used cookie IDs to request collected data. This approach is similar to the symmetric approach in Section IV. [67] reported that some companies requested ID cards or affidavits, while others directly used the cookie IDs in the browser. Neither approach proves that the requestor is really the consumer about whom the data was collected.

Kröger [57] studied mobile applications and observed an even more fragile ecosystem with discontinued apps and disappearing consumer accounts while processing SARs. Another conclusion of this study was to move away from email-initiated and manual processes which are prone to errors. In terms of compliance, the analysis by Herrmann and Lindemann [53] showed 43% compliance with access requests vs. 57% compliance with deletion requests.

In addition to the above, Dabrowski et al. [46] investigated cookie usage and how it is affected by privacy regulations, ngreporti that 11% of EU-related websites set cookies for US-based consumers, though not for EU-based ones. Furthermore, up to 46.7% of websites that appear in both the 2016 and 2018 Alexa top 100,000 sites stopped using persistent cookies without consumer permission. In the standardization realm, Zimmeck and Alicki [73] focused on "Do Not Sell" requests which informs the websites that they may not share the consumer's information with third parties. [73] developed a browser extension (OptMeowt) which conveys "Do Not Sell" requests to the websites through headers and cookies.

Asymmetric access tokens. As mentioned, cookies are notoriously known for being used in attacks (e.g., session hijacking) due to their symmetric nature [36], [50], [42], [19].

The most similar work to VICEROY from a technical perspective is Origin Bound Certificates (OBCs) [48] (also see RFC 8471 [29]), which aims to strengthen TLS client authentication by converting cookies to asymmetric access tokens. In OBC, the client generates a unique self-signed TLS client certificate for each website, in order to remain unlinkable across websites. Although this does not authenticate the client to the website (due to the self-signed certificate), it does allow the server to ascertain whether this is the *same* client from a previous interaction. One benefit of this is that cookies can be bound to an OBC, such that, even if stolen, they cannot be used by an adversary. This is very similar to how we bind wrappers to a client-generated key. One key difference is that, in OBC, the cookies themselves are modified, whereas our use of wrappers means that VICEROY can be incrementally deployed on top of existing systems without needing to modify how they use cookies. Another difference is that per-site certificates would not be suitable for accountless consumers, as these would allow servers to link together different visits from the same client. The alternative of generating per-TLS-session certificates introduces the key explosion problem, which we address in Section IV. Finally, OBC and VICEROY differ in terms of their primary objectives: OBC aims to strengthen TLS channels in general, but is thus tightly coupled to the TLS protocol, whereas VICEROY aims to provide a specific mechanism for supporting VCRs, which can be run over any communication protocol.

Another technology related to asymmetric access tokens is the FIDO Universal Authentication Framework (UAF) [11]. UAF allows users to authenticate to servers using mechanisms other than passwords, e.g., biometrics. It also supports multi-factor authentication, e.g., requiring both a PIN and a biometric. The devices used to obtain such factors are called *authenticators*. During registration, authenticators generate and register a server-specific *authentication key*, which they then use for subsequent authentications. Whilst similar to the approach used in VICEROY, there are several reasons why the FIDO UAF protocol is not directly suitable for VICEROY. First, VICEROY requires a fresh key pair to be generated for every *session* per website, since these keys will be paired with session-specific cookies. FIDO UAF does not meet this requirement, since only one key pair is generated per website. Even if the FIDO UAF protocol were modified to generate a key pair per session per website, this would lead to the key explosion problem described in Section IV-C1, and it would be particularly challenging to store all these keys in a resource-constrained device, such as the Solokey. In contrast, in VICEROY, the use of BIP32 is critical to handle the significantly larger volume of keys and facilitate implementation on resource-constrained devices. Second, using FIDO UAF would require trusted devices to be online when generating cookie wrappers, since both the public and private keys are generated by the trusted device. This is not ideal for security conscious users who might prefer to keep their trusted device offline most of the time (e.g., in a locked safe). In contrast, VICEROY supports both casual and security-conscious use cases by not requiring the trusted device to be present when generating public keys for cookie wrappers.

IX. Conclusions & Future Work

Motivated by recent GDPR and CCPA regulations granting (even) accountless consumers rights to access to data gathered about their behavior by web servers, we construct and evaluate VICEROY, a framework for authenticating accountless consumers. VICEROY is secure with respect to malicious clients and HbC servers, easy to deploy, and imposes fairly low overhead. Natural directions for future work include: (1) integration with client-side trusted execution environments (TEEs), (2) more extensive support for the MODIFY VCR type, and (3) support for unilateral server deletion of accountless-consumer data, which can occur if a server decides to delete consumer data without an explicit request.

Acknowledgements

We thank NDSS reviewers for their valuable comments which helped improve this paper. UCI authors were supported in part by funding from NSF Awards SATC-1956393 and CICI-1840197. The second author was also supported in part by The Nakajima Foundation.

REFERENCES

[1] "Android keystore system," https://developer.android.com/training/articles/keystore, accessed: 2021-04-20.

[2] "Anonymous Submission NDSS'23," https://www.dropbox.com/sh/866gwbud2x4fuaq/AAD5FJnmKuL3at5O1Zy8mZyFa?dl=0.

[3] "BIP 0032," https://en.bitcoin.it/wiki/BIP_0032, accessed: 2021-04-05.

[4] "BIP32," https://github.com/bitcoinjs/bip32, accessed: 2021-04-01.

[5] "bip32," https://github.com/sammyne/bip32, accessed: 2021-04-20.

[6] "BIP32," https://github.com/NovaCrypto/BIP32, accessed: 2021-04-20.

[7] "BIP32 Implementation using Python," https://github.com/ismailakkila/bip32, accessed: 2021-04-20.

[8] "Browserify," https://github.com/browserify/browserify, accessed: 2021-04-01.

[9] "Cookie Matching," https://developers.google.com/authorized-buyers/rtb/cookie-guide, accessed: 2021-04-26.

[10] "Express," https://expressjs.com/, accessed: 2021-05-01.

[11] "FIDO UAF," https://fidoalliance.org/specs/fido-uaf-v1.2-ps-20201020/fido-uaf-overview-v1.2-ps-20201020.html.

[12] "Google Analytics," https://developers.google.com/analytics/devguides/collection/analyticsjs/cookie-usage, accessed: 2021-02-18.

[13] "Hierarchical Deterministic Wallets," https://github.com/marctrem/BIP32c, accessed: 2021-04-20.

[14] "HTTPS encryption on the web," https://transparencyreport.google.com/https/overview.

[15] "Intel SGX DCAP," https://01.org/intel-softwareguard-extensions/downloads/intel-sgx-dcap-1.6-release.

[16] "Intel SGX EPID Attestation," https://api.portal.trustedservices.intel.com/EPID-attestation.

[17] "Intel® Software Guard Extensions," https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html, accessed: 2021-04-20.

[18] "Ledger," https://www.ledger.com/, accessed: 2021-04-20.

[19] "MITRE ATT&CK: Steal Web Session Cookie," https://attack.mitre.org/techniques/T1539/.

[20] "native-messaging," https://www.npmjs.com/package/native-messaging, accessed: 2021-05-01.

[21] "Native Messaging Protocol," https://developer.chrome.com/docs/apps/nativeMessaging/\#native-messaging-host-protocol, accessed: 2021-05-01.

[22] "Node.js," https://nodejs.org/en/, accessed: 2021-02-18.

[23] "NPM uuid," https://www.npmjs.com/package/uuid, accessed: 2021-02-18.

[24] "Solokeys," https://solokeys.com/.

[25] "Solokeys Code base," https://github.com/solokeys/solo.

[26] "Solokeys Hardware schematics," https://github.com/solokeys/solo-hw.

[27] "Storing Keys in the Secure Enclave," https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/storing_keys_in_the_secure_enclave, accessed: 2021-04-20.

[28] "Tamarin Prover," https://tamarin-prover.github.io/.

[29] "The Token Binding Protocol Version 1.0," https://tools.ietf.org/html/rfc8471, accessed: 2021-04-26.

[30] "tiny-secp256k1," https://www.npmjs.com/package/tiny-secp256k1?activeTab=versions, accessed: 2021-05-01.

[31] "Tor Project," https://www.torproject.org/.

[32] "Trezor Firmware," https://github.com/trezor/trezor-firmware/.

[33] "What Technology Change Enables 1 Terabyte (TB) Enclave Page Cache (EPC) size in 3rd Generation Intel® Xeon® Scalable Processor Platforms?" https://www.intel.com/content/www/us/en/support/articles/000059614/software/intel-security-products.html.

[34] "YubiKey," https://www.yubico.com/.

[35] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The Web Never Forgets: Persistent Tracking Mechanisms in the Wild," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 674–689.

[36] Z. A. Alizai, H. Tahir, M. H. Murtaza, S. Tahir, and K. Mcdonald-Maier, "Key-based cookie-less session management framework for application layer security," *IEEE Access*, vol. 7, pp. 128 544–128 554, 2019.

[37] P.-L. Aublin, F. Kelbert, D. O'Keeffe, D. Muthukumaran, C. Priebe, J. Lind, R. Krahn, C. Fetzer, D. Eyers, and P. Pietzuch, "Libseal: Revealing service integrity violations using trusted execution," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18, 2018. [Online]. Available: https://doi.org/10.1145/3190508.3190547

[38] J. Ausloos and P. Dewitte, "Shattering One-Way Mirrors. Data Subject Access Rights in Practice," *Data Subject Access Rights in Practice (January 20, 2018). International Data Privacy Law*, vol. 8, no. 1, pp. 4–28, 2018.

[39] C. Boniface, I. Fouad, N. Bielova, C. Lauradoux, and C. Santos, "Security Analysis of Subject Access Request Procedures," in *Annual Privacy Forum*. Springer, 2019, pp. 182–209.

[40] M. Cagnazzo, T. Holz, and N. Pohlmann, "GDPiRated – Stealing Personal Information On- and Offline," in *European Symposium on Research in Computer Security*. Springer, 2019, pp. 367–386.

[41] P. Calvano, "An Analysis of Cookie Sizes on the Web," https://paulcalvano.com/2020-07-13-an-analysis-of-cookie-sizes-on-the-web/.

[42] S. Calzavara, A. Rabitti, and M. Bugliesi, "Sub-session hijacking on the web: Root causes and prevention," *Journal of Computer Security*, vol. 27, no. 2, pp. 233–257, 2019.

[43] C. Castelluccia, L. Olejnik, and T. Minh-Dung, "Selling Off Privacy at Auction," in *Network and Distributed System Security Symposium (NDSS)*. San Diego, California, United States: ISOC, Nov. 2014.

[44] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Proceedings of IEEE 36th Annual Foundations of Computer Science*. IEEE, 1995, pp. 41–50.

[45] K. Crichton, N. Christin, and L. F. Cranor, "How do home computer users browse the web?" *ACM Trans. Web*, vol. 16, no. 1, sep 2021. [Online]. Available: https://doi.org/10.1145/3473343

[46] A. Dabrowski, G. Merzdovnik, J. Ullrich, G. Sendera, and E. Weippl, "Measuring Cookies and Web Privacy in a Post-GDPR World," in *International Conference on Passive and Active Network Measurement*. Springer, 2019, pp. 258–270.

[47] M. Di Martino, P. Robyns, W. Weyts, P. Quax, W. Lamotte, and K. Andries, "Personal Information Leakage by Abusing the {GDPR}'Right of Access'," in *Fifteenth Symposium on Usable Privacy and Security ({SOUPS} 2019)*, 2019.

[48] M. Dietz, A. Czeskis, D. Balfanz, and D. S. Wallach, "Origin-Bound Certificates: A Fresh Approach to Strong Client Authentication for the Web," in *21st {USENIX} Security Symposium ({USENIX} Security 12)*, 2012, pp. 317–331.

[49] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz, "Measuring HTTPS adoption on the web," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1323–1338. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/felt

[50] O. B. Fredj, "Spheres: An efficient server-side web application protection system," *International Journal of Information and Computer Security*, vol. 11, no. 1, pp. 33–60, 2019.

[51] C. A. General, "California Consumer Privacy Act Regulations," https://oag.ca.gov/sites/all/files/agweb/pdfs/privacy/oal-sub-final-text-of-regs.pdf, 2020.

[52] R. Gonzalez, L. Jiang, M. Ahmed, M. Marciel, R. Cuevas, H. Metwalley, and S. Niccolini, "The cookie recipe: Untangling the use of cookies in the wild," in *2017 Network Traffic Measurement and Analysis Conference (TMA)*, 2017, pp. 1–9.

[53] D. Herrmann and J. Lindemann, "Obtaining personal data and asking for erasure: Do app vendors and website owners honour your privacy rights?" *arXiv preprint arXiv:1602.01804*, 2016.

[54] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9000

[55] S. Jordan, "A Comparison of Notice and Consent Requirements under the GDPR, the CCPA/CPRA, and the FCC Broadband Privacy Order," 2021.

[56] S. Jordan, S. Narasimhan, and J. Hong, "Collection, Use, and Sharing of Personal Information," 2021.

[57] J. L. Kröger, "Subject Access Request response data - 105 iOS and 120 Android apps," 2020. [Online]. Available: http://dx.doi.org/10.14279/depositonce-10338

[58] C. Legislature, "California Consumer Privacy Act of 2018 (as amended by the California Privacy Rights Act of 2020)," https://www.oag.ca.gov/privacy/ccpa, 2020.

[59] J. R. Mayer and J. C. Mitchell, "Third-Party Web Tracking: Policy and Technology," in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 413–427.

[60] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *International conference on computer aided verification*, 2013.

[61] Y. Nakatsuka, E. Ozturk, A. Paverd, and G. Tsudik, "CACTI: Captcha Avoidance via Client-side TEE Integration," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, Aug. 2021.

[62] E. Parliament and Council, "General Data Protection Regulation, Regulation (EU) 2016/679 (as amended)," https://eur-lex.europa.eu/eli/reg/2016/679/2016-05-04, 2016.

[63] A. Paverd, A. Martin, and I. Brown, "Modelling and automatically analysing privacy properties for honest-but-curious adversaries," *Tech. Rep*, 2014.

[64] J. Pavur and C. Knerr, "GDPArrrrr: Using Privacy Laws to Steal Identities," *arXiv preprint arXiv:1912.00731*, 2019.

[65] E. Rescorla, H. Tschofenig, and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," RFC 9147, Apr. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9147

[66] T. Urban, M. Degeling, T. Holz, and N. Pohlmann, ""Your hashed IP address: Ubuntu." perspectives on transparency tools for online advertising," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 702–717.

[67] T. Urban, D. Tatang, M. Degeling, T. Holz, and N. Pohlmann, "A Study on Subject Data Access in Online Advertising After the GDPR," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2019, pp. 61–79.

[68] M. Veeningen, B. d. Weger, and N. Zannone, "Modeling identity-related properties and their privacy strength," in *International Workshop on Formal Aspects in Security and Trust*. Springer, 2010, pp. 126–140.

[69] ——, "Formal privacy analysis of communication protocols for identity management," in *International Conference on Information Systems Security*. Springer, 2011, pp. 235–249.

[70] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: Using Hard AI Problems for Security," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2003, pp. 294–311.

[71] P. Wuille, "BIP32 – Security Implications," https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki#Implications.

[72] Z. Zhang, X. Ding, G. Tsudik, J. Cui, and Z. Li, "Presence Attestation: The Missing Link in Dynamic Trust Bootstrapping," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, 2017. [Online]. Available: https://doi.org/10.1145/3133956.3134094

[73] S. Zimmeck and K. Alicki, "Standardizing and Implementing Do Not Sell," in *Proceedings of the 19th Workshop on Privacy in the Electronic Society*, 2020, pp. 15–20.

X. Formal Protocol Specification

Listing 1 presents the formal model of the VICEROY protocol and the security properties verified using the Tamarin prover [60]. For details of the Tamarin syntax and conventions, please refer to the Tamarin prover website [28].

```
theory VICEROY
begin

builtins: signing, hashing

// Public key infrastructure
rule Register_pk:
  [ Fr(~skA) ]
  --[Unique($A)]->
  [ !Sk($A, ~skA), !Pk($A, pk(~skA)), Out(pk(~skA)) ]

rule Reveal_sk:
  [ !Sk(A, skA) ] --[ RevSk(A) ]-> [ Out(skA) ]


// Derivable asymmetric keys
rule Register_derived_key:
  [ !Sk($A, skA), Fr(~dskA) ]
  -->
  [ !Dsk($A, ~dskA), !Dpk($A, pk(~dskA)), Out(pk(~dskA)) ]

rule Reveal_dsk:
  [ !Dsk(A, dskA) ] --[ RevDsk(A) ]-> [ Out(dskA) ]


/* We formalize the following protocol

Trusted device (T)  |  Client device (C)  |  Web server (S)

[Optional] Trusted device generates master private key sk(t) and sends device public key pk(t/i) to the
    client's device [not included in the model since it is not mandatory to use a separate trusted device].

When the client begins interacting with a website, the website issues the client with a cookie.
0. S -> C: cookie

The client derives a fresh VCR public key pk(t/i/j) from the device public key and sends it to the server
    along with the cookie.
1. C -> S: cookie, pk(t/i/j)

The server returns a cookie wrapper, which is a signature over the cookie and the provided VCR public key.
2. S -> C: sign_{sk(S)}{h(cookie, pk(t/i/j))}

When issuing a VCR, the user signs the request and cookie using the corresponding private key on their
    trusted device [not included in this model], and then sends the cookie, request, public key, wrapper, and
     signature to the server.
3. C -> S: cookie, request, pk(t/i), sign_{sk(S)}{h(cookie, pk(t/i/j))}, sign_{sk(t/i/j)}{h(request, cookie)}

If all the signatures can be verified, the server accepts the VCR and performs the requested operation. */

rule S_0:
    [ Fr(~cookie) ]
  --[  ]->
    [ Out( ~cookie ), !State_S_0($S, ~cookie) ]

rule C_1:
  let dpkC = pk(~dskC)
      m1 = <cookie, dpkC>
  in
    [ In(cookie), Fr(~dskC) ]
  --[ Request_wrapper( dpkC ) ]->
    [ Out( m1 ), State_C_1(~dskC, dpkC, cookie) ]

rule S_1:
  let m1 = <cookie, dpkC>
      m2 = sign(h(<cookie, dpkC>), skS)
  in
    [ !State_S_0($S, cookie), !Pk($S, pkS), !Sk($S, skS), In( m1 ) ]
  --[ Issue_wrapper($S, dpkC, <cookie>) ]->
    [ Out( m2 ), !State_S_1($S, pkS) ]
```

```
rule C_2:
  let request = <'op', ~nonce, pkS>
      m3 = <cookie, request, dpkC, wrapper, sign( h(<request, cookie>), dskC ) >
  in
    [ State_C_1(dskC, dpkC, cookie), !Pk($S, pkS), In(wrapper), Fr(~nonce) ]
  --[ Eq( verify(wrapper, h(<cookie, dpkC>), pkS), true )
    , Issue_VCR(dskC, $S, <cookie, request>) ]->
    [ Out( m3 ) ]

rule S_2:
  let request = <'op', nonce, pkS>
      m3 = <cookie, request, dpkC, wrapper, client_sig>
  in
    [ !State_S_1($S, pkS), In( m3 ) ]
  --[ Eq( verify(wrapper, h(<cookie, dpkC>), pkS), true )
    , Eq( verify(client_sig, h(<request, cookie>), dpkC), true )
    , Unique(nonce)
    , Accept_VCR($S, dpkC, <cookie, request>) ]->
    [  ]

restriction Equality:
  "All x y #i. Eq(x,y) @i ==> x = y"

restriction Uniqueness:
   "All x #i #j. Unique(x) @ i & Unique(x) @ j ==> #i = #j"

/* Wrapper unforgeability: whenever the server accepts a VCR from a client, then that server had previously
    issued a cookie wrapper to that client for the same cookie, or the adversary performed a long-term key
    reveal on the server, or the adversary knows the client's derived private key. */
lemma wrapper_unforgeability:
  " All server dskC cookie request #i.
       Accept_VCR(server, pk(dskC), <cookie, request>) @ i
     ==>
       (Ex #j. Issue_wrapper(server, pk(dskC), <cookie>) @ j & j < i)
     | (Ex #r. RevSk(server) @ r) | (Ex #r. KU(dskC) @ r) "

/* VCR unforgeability: whenever the server accepts a VCR, then the client with the corresponding private key
    issued that VCR, or the adversary performed a long-term key reveal on the server, or the adversary knows
    the client's derived private key. */
lemma VCR_unforgeability:
  " All server dskC cookie request #i.
       Accept_VCR(server, pk(dskC), <cookie, request>) @ i
     ==>
       (Ex #j. Issue_VCR(dskC, server, <cookie, request>) @ j & j < i)
     | (Ex #r. RevSk(server) @ r) | (Ex #r. KU(dskC) @ r) "

/* Replay resistance: the server will not accept a VCR for the same cookie and request combination more than
    once, unless the adversary knows the client's derived private key. */
lemma replay_resistance:
  " All server dskC cookie request #i #j.
       Accept_VCR(server, pk(dskC), <cookie, request>) @ i &
       Accept_VCR(server, pk(dskC), <cookie, request>) @ j
     ==>
       #i = #j | (Ex #r. KU(dskC) @ r) "

/* Consistency check: the server can accept a VCR without the adversary having performed a long-term key
    reveal on the server or knowing the client's derived private key. */
lemma accept_vcr_possible:
  exists-trace
  " Ex server dskC params #i.
      Accept_VCR(server, pk(dskC), params) @ i
    & not (Ex #r. RevSk(server) @ r)
    & not (Ex #r. KU(dskC) @ r) "

end
```

Listing 1. Tamarin specification of the messages exchanged in VICEROY