Bootstrapping Online Trust: Timeline Activity Proofs

Constantin Cătălin Drăgan and Mark Manulis

Surrey Centre for Cyber Security, University of Surrey, UK
c.dragan@surrey.ac.uk, mark@manulis.eu

Abstract. Establishing initial trust between a new user and an online service, is being generally facilitated by centralized social media platforms, i.e., Facebook, Google, by allowing users to use their social profiles to prove "trustworthiness" to a new service which has some verification policy with regard to the information that it retrieves from the profiles. Typically, only static information, e.g., name, age, contact details, number of friends, are being used to establish the initial trust. However, such information provides only weak trust guarantees, as (malicious) users can trivially create new profiles and populate them with static data fast to convince the new service.

We argue that the way the profiles are used over (longer) periods of time should play a more prominent role in the initial trust establishment. Intuitively, verification policies, in addition to static data, could check whether profiles are being used on a regular basis and have a convincing footprint of activities over various periods of time to be perceived as more trustworthy.

In this paper, we introduce *Timeline Activity Proofs* (TAP) as a new trust factor. TAP allows online users to manage their timeline activities in a privacy-preserving way and use them to bootstrap online trust, e.g., as part of registration to a new service. In our model we do not rely on any centralized social media platform. Instead, users are given full control over the activities that they wish to use as part of TAP proofs. A distributed public ledger is used to provide the crucial integrity guarantees, i.e., that activities cannot be tampered with retrospectively. Our TAP construction adopts standard cryptographic techniques to enable authorized access to encrypted activities of a user for the purpose of policy verification and is proven to provide *data confidentiality* protecting the privacy of user's activities and *authenticated policy compliance* protecting verifiers from users who cannot show the required footprint of past activities.

1 Introduction

Social interactions have always been guided by the trust between the parties involved. The problem with setting initial trust arises when there exists no pre-established relationship between the entities. In this case they need to trust some third party to facilitate the initial communication.

In the online environment, many existing social media platforms, e.g., Facebook, Google, LinkedIn, act as identity providers for their users and offer what is called a *social login* service. This service, often realised based on the Open ID Connect framework [14], is widely used as a trust anchor upon the initial registration of a new user to an online service, and can also be used for subsequent authentication procedures. As part of the social login, a user can authorize other parties, e.g., mobile applications, online services, to access data kept in the user's social profile such as contact information, number of connections, posts, comments, photographs, etc.

At the moment, upon registration, many applications and online services are using only static data such as the user's name, age, contact details, total number of connections/friends, etc. The initial trust is thus established through a limited snapshot of the profile. This, however, offers only weak trust guarantees. Due to the ease of setting up social profiles, only little time would be needed to setup a "fake" profile and populate it with snapshot data to satisfy the checks that an online service performs on new users. The main reason is that such snapshots lack the historic perspective and cannot be used to decide whether a profile has been used frequently over a longer period of time.

One way to reduce the possibility of creating fake profiles fast is to consider the longevity and frequency of the profile's online interaction. This can be done by looking into the user's *timeline activities* such as posts, comments, photographs, and other interactions, and using them as an additional trust factor. There are, however, a number of privacy challenges associated with timeline activities in existing centralized social networks. First, users typically do not have full control over which activities can be accessed by authorised third parties. The access is defined by the settings of the social login provider and is often based on the "all-or-nothing" sharing approach. Users would be forced to directly modify their timeline, e.g., remove some of their activities, to be able to restrict what type of information they wish to release to third parties. More importantly, users are not aware of the amount of information that is stored or shared by the social service provider. After the Cambridge Analytica scandal [11], Facebook offered a better understanding about the data they collect [17], i.e., the received and declined friend requests, entire conversations or files exchanged via the Facebook messenger, and history of calls and messages on mobile phones. There is also the lack of transparency over the information which third parties acquire from the social service provider [16].

The above shows a more general problem with centralized social services for using timeline activities to facilitate online trust establishment, namely the need to trust the social service provider to protect the privacy of user's activities. The lack of trust naturally leads to the following question: *Can timeline activities be fully controlled by the users (without reliance on any centralized service provider) and used as an additional trust factor in online interactions?*

The main challenge behind putting users in full control of their activities is the integrity. Note that in order to be used as a trust factor that reduces the risk of fast creation of fake profiles it is important to guarantee that activities cannot be tampered with. In case of centralized social services, the integrity of the timeline activities is guaranteed by the social service provider and in most cases users are not allowed to introduce new activities retrospectively, but can still remove them if needed. A decentralized system where timeline activities are fully controlled by the users would need to achieve similar guarantees with respect to their integrity.

Distributed public ledgers can be a solution, such that users can keep footprints of their timeline activities, and authorize third parties to access them. The integrity of activities would be ensured by the properties of the ledger that acts as an append-only list, and where previous activities can be updated by introducing new activities that would be linked to them. The ability of users to remove an activity depends on how the data is stored and how it is linked with the ledger. We anticipate a hybrid approach, where sensitive data will be encrypted and stored in an external database, such that hashes to this encrypted data are committed to the public ledger.

Such a system would enable users to authorize third parties to access their activities at any level of granularity and over different periods of time, thus offering higher flexibility to define verification policies.

Our Contribution. In this paper we propose *Timeline Activity Proofs* (TAP) allowing users to establish trust with online parties based on user's activities and without relying on any centralized service provider. TAP can be used as a building block that adds a new trust factor — timeline activities — to an online trust establishment process. TAP keeps the user's timeline activities by storing pairs of (time, entry) in a distributed public ledger, with the user computing the entry from the activity, and the ledger appending a timestamp prior to recording it. A stored entry can contain public descriptors of the activity, i.e., name and additional keywords (tags), and a ciphertext of the actual activity data. Furthermore, the entry is linked to the user who owns the activity via a digital signature. TAP uses symmetric encryption to preserve privacy of the activity data. A dedicated key management ensures that independent symmetric keys are pseudorandomly derived for each activity record. Such key derivation approach also enables granular disclosure of activities to verifiers as part of the proof protocol that is performed between the user and the verifier on input of a public activity-based verification policy. For the proofs in TAP we define two security properties: *data confidentiality*, an indistinguishability property protecting privacy of the activity data from unauthorised access, and *authenticated policy compliance*, a soundness property ensuring that only legitimate owners of activities can use them for proofs and that verification policies can be passed only if the user has committed suitable activities that can satisfy them into the public ledger.

Related Work. Prior works have also focused on ensuring privacy in social networks, by following two main directions in eliminating the need for a central service provider. The first by distributing the functionalities of the social network, e.g., Safebook [5] that provides a distributed privacy-preserving social media platform. The second by realizing diverse functionalities of the centralized social network in a distributed manner using cryptographic techniques [1, 9]. Both of these approaches (and specifically their realization) are focused on achieving privacy for the user and his data, and less on the user's behavior. This has integrity implication for ensuring trust, as users can alter at any time their activity and make any claim they wish, i.e., in Safebook [5] integrity means protection against unauthorized tampering, with the user being authorized at any time to change his data.

A concept for establishing initial trust based on activities is the approach used by China's Social Credit System, called Sesame Credit [13]. There, the social activities of an individual are used to compute a "social score" that is used as a trust factor in the online environment. However, this is done without any privacy and under the control of a central authority where the user has no control of which social activities are stored.

2 Modeling Timeline Activity Proofs

2.1 Entities and their roles

Users. Each *user* owns a digital identity, that we model by using a secret-public key pair $(\mathsf{sk}, \mathsf{pk})$. The public key pk is used as the public identifier in our system, and any statement made by a user will be accompanied by this public key. They can create multiple digital identities by generating multiple secret-public key pairs.

Users are in charge of submitting their online activities, and can do so with any of the digital identities they posses. Furthermore, they have full control over which activities would be submitted, and what information should they include.

Users create activities that are stored in a ledger. Later, they would be used to prove statements about the users that submitted them. These activities are bound to users via public keys and appropriate authenticators. When an activity is added to the ledger, a timestamp with the time of the submission is appended to it; the user has no control over this time. Later, this time can be used as historic evidence of users performing these actions at a particular time.

In practice, users can be ordinary people, businesses (online shops, services, etc.), or any other organizations with an online presence.

Public Ledger. We use a distributed public ledger to maintain a list of activities. The ledger is assumed trusted, and offers protection for the integrity of the data. This core functionality of the ledger is modeled by considering an *append-only list*. Moreover, the types of entries that can be introduced are restricted to a clear entry format that has to be respected, via a validation mechanism for the data in the entry.

The ledger is tasked with maintaining the timeline for the data. More precisely, the ledger will have a *tamper-proof time mechanism*, and add timestamps to each entry. The method for registering time to each entry is simplified by assuming there exists no delay between the time an entry is received and the time it gets added.

In its essence a public ledger is just a secure append-only database. This narrow view of the storage format, allows for more complex search queries. One can define a search mechanism that considers the certain constraints on the information from the entry and the time it was added. Then, apply this search to all entries in an efficient manner, and extract only those that satisfy the search condition.

Formally, the public ledger $\mathsf{PL}(\mathsf{data}, \mathsf{val}, \mathsf{clock}) = (\mathsf{Setup}, (\mathsf{GetTime}, \mathsf{GetInterval}), \mathsf{Append}, \mathsf{Search})$ consists of an information type data, an evaluation mechanism val for the validity of the information, an internal clock, and the following algorithms:

- $\mathsf{Setup}(\lambda) : \mathsf{pp}$. Given as input the security parameter λ, it generates a list of public parameters pp that contains *an empty append-only list* for storing entries. In addition, it initializes the clock.
- $(\mathsf{GetTime}, \mathsf{GetInterval}) : (\mathsf{time}, i)$. $\mathsf{GetTime}()$ outputs the current time value of clock. $\mathsf{GetInterval}(\mathsf{time})$ returns the interval i for a given time.
- $\mathsf{Append}(\mathsf{data}) : (\mathsf{time}, \mathsf{data}) \cup \{\perp\}$. Given some data as input, it returns either an entry $(\mathsf{time}, \mathsf{data})$ that has been registered in the append-only list, or an error symbol $\perp$. First, it evaluates data based on the val mechanism, and aborts with $\perp$ if it fails. Then, it calls $\mathsf{time} \leftarrow \mathsf{GetTime}()$, creates an entry $(\mathsf{time}, \mathsf{data})$ and

appends it to the append-only list. If this last operation succeeds, it returns the entry, otherwise it returns the same error symbol $\perp$.

- $\mathsf{Search}(Q) : L$. Given a search query Q, it returns a (possibly empty) list of valid entries L from the ledger that satisfy the search conditions. The search query Q can depend on the information in data and its time.

Remark 1 (Instantiations). There are a number of existing implementations for distributed public ledgers with the most notable being *Bitcoin* [12] and *Ethereum* [18]. Both offer a secure timestamping system that records the time when each entry was processed. The search functionality is not native to these systems, but recent results, i.e., smartbit [15], do offer a method for searching. However, these two solutions impose a limitation on the size of the data that can be stored.

Public Ledger with external database. In general, a public ledger offers an idealized approach for the integrity of data, with no method of data removal, even for the user who submitted that data. In practice, one would like to consider scenarios where users would want to remove their activity data. In light of this, it is natural to apply a hybrid approach with the data being stored in a separate (distributed) database, and use periodic commitments, similar to the approach in [7] suggested for decentralized anonymous credentials. The link between the records in the database and the entries in the ledger can be maintained by a unique id $\mathsf{rec_{id}}$ of an activity record. We can extend the definition of public ledgers to consider an external (distributed) database with records of the form $(\mathsf{rec_{id}}, \mathsf{data})$, and adapt the format for the entries in the ledger to consider commitments to that record: $(\mathsf{time}, \mathsf{rec_{id}}, \mathsf{H}(\mathsf{data}, \mathsf{rec_{id}}))$, where H is a cryptographic hash function. The following changes to the algorithms defined for the public ledger would account for the use of the database: Setup, in addition to initialization of the ledger, would also initialize the database; Append would record the data in the database, prior to adding its commitment to the ledger; and Search would first look in the database, and then filter invalid records based on the commitments in the ledger. The following algorithm would then allow users to remove their activity data from the database:

- $\mathsf{Remove}(\mathsf{data}) : \mathsf{rec_{id}} \cup \{\perp\}$. If the record $(\mathsf{rec_{id}}, \mathsf{data})$ has been removed from the database it returns $\mathsf{rec_{id}}$; otherwise it returns the error symbol $\perp$.

Remark 2 (Time Delays). The time a data entry is recorded in the ledger, can be used to construct a correct timeline that later can be used to evaluate statements about the user. Due to how current ledgers are implemented there exists a delay between the time a transaction is sent and the time it is added to the ledger, even more so in the case of hybrid approaches. This time difference may lead to situations where statements that should hold (based on the time they were submitted) would not.

One can deal with this situation by first defining a fixed bound on the delay that one may expect when submitting entries to the ledger. Then, either relax the verification policy to consider time intervals for activities (that may depend on this delay) instead of a fixed time, or consider two timestamps with one provided by the user and an other introduced by the ledger. For the latter solution, we can include verification over the difference between these two timestamps w.r.t the expected delay.

Verifiers. *Verifiers* can define policies over timeline activities. Through interaction with a user a verifier can check whether this user satisfies their policy based on the user's activities submitted to the ledger. First, the verifier extracts from the ledger records that they deem relevant for their verification, i.e., based on the type of activity, the time it was submitted, and user's public key. Then, they request authorization from the user to open these entries and assert if they satisfy the verifier's policy.

2.2 Modeling Timeline Activities

Timed Activities. Activities model online actions performed by a user. They serve as historic evidence of trust, and are defined based on the following template:

$$\langle \mathsf{pk}, \mathsf{atime}, \mathsf{aname}, \mathsf{count}, \mathsf{adata}, [\mathsf{tags}] \rangle$$

- pk is the public identity of the user who created this activity;
- atime is the time the activity was submitted;
- aname describes the type of activity created, e.g. *photo, address, email.* They are predefined and each user can select only one type for each activity.
- count identifies an activity among other activities of the same type submitted within the same time interval;
- adata is the actual activity data. This information is never added directly to the ledger, only an encrypted version, denoted cdata, is added;
- $[\mathsf{tags}]$ is an optional field that introduces additional information not captured by the name of activity. It is a set of name-value pairs $(\mathsf{tname}, \mathsf{tvalue})$, e.g., a photo may contain the tags $\{(people, \text{John}), (location, \text{New York}), (date, \text{2017-01-01})\}$.

Remark 3. In our TAP construction the tags remain in clear. This feature allows verifiers to search more efficiently and identify which entries in the ledger could be used for verification. Further extensions may include symmetric searchable encryption techniques, e.g., [4], to additionally protect the privacy of the tags.

The activities must follow a predefined public format that contains the set of allowed activity names $\mathsf{aname} \in \mathsf{ANames}$, and the set of optional permitted tag names $\mathsf{tname} \in \mathsf{TNames}$. Such format is needed for independent formulation of verification policies.

Each activity can uniquely be identified by a tuple $(\mathsf{pk}, \mathsf{atime}, \mathsf{aname}, \mathsf{count})$, with count being a unique counter value assigned to activities of the name type aname in the time interval $\mathsf{GetInterval}(\mathsf{atime})$, for the user pk.

Activities can be separated based on the number of occurrences of the same type that the user can create:

- *static.* This type of activity should appear once, e.g., date of birth, name at birth, unique social security number, etc.
- *dynamic.* There is no restriction on the number of activities of this type, e.g., posting pictures on Facebook, change of email or post address, etc.

Policies and verification of Timeline Activities. Activities are submitted with the purpose of building some historic evidence that would later be used to prove certain statements about their owners. This verification is modeled using a *policy* Ψ over a set of activities A, and outputs whether they satisfy Ψ, i.e., $\Psi(\mathsf{A}) = 1$. The policy Ψ is not specific to any particular user, and doesn't use the pk component of the activity.

We consider two categories of policies based on the information required from the input activities:

- *meta-data policy*, where only information from the public components of the activities, i.e., the name, time, counter, and tags, used to evaluate the policy; and
- *data-dependent policy*, where information from the actual data of the activities is required, in addition to some of the meta-data.

The former type of policy can be easily evaluated by verifiers regardless the information contained in the data. While the latter can only be evaluated with an evaluation mechanism over that activity data, i.e., directly checking whether a photograph contains certain elements/keywords. We assume the policy includes such mechanism.

Example 1. Consider an online method that verifies the documents needed for opening a bank account, before setting an in-person interview. Some of the requirements include a valid identification document, i.e., passport or id card, and the address with a proof of living there for the past 3 months. The user pk_U that applies to this verification on May 2017, has the following activities stored in the ledger that could be used.

$$< \mathsf{pk}_U, \text{2017-01-01T01:01:01}, \text{id card}, \ldots >$$
$$< \mathsf{pk}_U, \text{2017-01-02T03:04:05}, \text{utility bill}, \ldots >$$
$$< \mathsf{pk}_U, \text{2017-02-03T04:05:06}, \text{utility bill}, \ldots >$$
$$< \mathsf{pk}_U, \text{2017-03-04T05:06:07}, \text{utility bill}, \ldots >$$

The verifier extracts these entries form the ledger by looking at the public key of the user, and the activity name and time. Then, interacts with the user to obtain the authorization needed to see the information from the id card and bills. If he can confirm the bills are under the user's real name (obtained from the id card), and that the address on those bills is the same as the one the user has provided, the verification succeeds (returns true) and the verifiers makes an appointment for this user.

2.3 Linking Timeline Activities with the Public Ledger

We map the abstract ledger to the type of information required to store user activities, and specify the condition that define validity of data. Entries in the ledger contain plain information like user, activity type, counter, the optional tags, and a ciphertext of the activity data. Every entry is also accompanied by an authenticator to certify that the entry has been submitted by the user pk. In our construction we use digital signatures as authenticators. The validity check ensures that the authenticator can be verified with the public key.

- $\mathsf{data} = (\mathsf{pk}, \mathsf{aname}, \mathsf{count}, \mathsf{cdata}, [\mathsf{tags}], \sigma)$ with pk the user public key, aname the type of activity, count the counter for the same type of activity, cdata the encrypted version of adata, $[\mathsf{tags}]$ the optional tags, and σ the authenticator that this submitted activity belongs to user pk.

- $\mathsf{val}(\mathsf{pk}, \mathsf{aname}, \mathsf{count}, \mathsf{cdata}, [\mathsf{tags}], \sigma)$ returns true if the authenticator σ verifies for the statement $(\mathsf{aname}, \mathsf{count}, \mathsf{cdata}, [\mathsf{tags}])$ using pk; otherwise it returns false.

To make the entries in the ledger consistent with the format of activities, described in Section 2.2, we view the components $(\mathsf{time}, \mathsf{data})$ of an entry as

$$(\mathsf{pk}, \mathsf{time}, \mathsf{aname}, \mathsf{count}, \mathsf{cdata}, [\mathsf{tags}], \sigma).$$

Typically, verifiers that evaluate a policy for users are assumed to already have the relevant inputs through some prior interaction with the user. Due to the outsourcing of data by the user to the ledger, we have a search mechanism allowing the policy verifier to extract needed information. We model the ledger as a database and use queries to search for entries that satisfy conditions used in the policy. These queries are generated based on the conditions the policy imposes over the activities. The conditions in the query can only look at the following components of an activity: public key, name, time, counter, and tags. The output of this type of search is more general than the one described by the conditions in the policy. The user is then tasked with providing a proof that a subset of those entries (that the verifier has found in the ledger) contain activities that satisfy the policy. We detail in the next section how our model of TAP realizes this.

Example 2. Consider the policy Ψ used in Example 1, and a public ledger that contains entries $(\mathsf{pk}, \mathsf{atime}, \mathsf{aname}, \mathsf{count}, \mathsf{cdata}, [\mathsf{tags}])$. We define our search query as a *select* command that looks for the latest entries that have the user's public key, an activity name in {id card, passport}, and 3 activities that point to a fixed address in the last 3 months. As the address can be realized by a number of different activities, we need to look for entries that contain { phone bill, utility bill, tax ...}.

2.4 Timeline Activity Proofs: Definition

Definition 1 (Timeline Activity Proofs). $\mathsf{TAP} = (\mathsf{Setup}, \mathsf{KGenU}, \mathsf{SubmitU}, \mathsf{IProofU}, \mathsf{IProofV})$ *with access to* $\mathsf{PL}(\mathsf{data}, \mathsf{val})$ *defined in Section 2.3, consists of the following algorithms:*

- $\mathsf{Setup}(\lambda) : \mathsf{pp}$. *This algorithm is run by a trusted third-party and generates all public parameters required by the system. This includes the predefined activity names* ANames*, and tag names* TNames*. Furthermore, it calls* $\mathsf{PL.Setup}$ *to initialize the ledger and create an empty append-only list, and start the clock.*
- $\mathsf{KGenU}(\mathsf{pp})$ *is a user run algorithm that returns either a valid secret-public key pair* $(\mathsf{sk}, \mathsf{pk})$*, or an error symbol* $\perp$ *to symbolize that it failed. It creates locally a secret-public key pair* $(\mathsf{sk}, \mathsf{pk})$*. The term secret key is used generically to contain all secret information that the user would require in the system, i.e. signing keys and the seed for activity based encryption key derivation. Moreover, it is possible for users to call this algorithm multiple times and register multiple public identities/pks that can be used in the system.*
- $\mathsf{SubmitU}(\mathsf{pp}, \mathsf{sk}, (\mathsf{aname}, \mathsf{adata}, [\mathsf{tags}]))$ *is an algorithm run by user* $(\mathsf{sk}, \mathsf{pk})$ *that wants to submit the activity* $(\mathsf{aname}, \mathsf{adata}, [\mathsf{tags}])$*, to the public ledger* PL*. It returns either an entry* $(\mathsf{pk}, \mathsf{atime}, \mathsf{aname}, \mathsf{count}, \mathsf{cdata}, [\mathsf{tags}], \sigma)$ *that has been successfully added to* PL*, or an error symbol* $\perp$ *if the submission failed. If required,*

there may exist some prior interaction with the ledger PL*, where the user could for instance synchronize the time or authenticate to the system. The user performs the following steps. First, he computes the counter* count *based on the activity name and time, an encryption* cdata *of* adata*, and the authenticator* σ*. Then, following an successful interaction with* PL *by calling* $\mathsf{Append}(\mathsf{pk}, \mathsf{aname}, \mathsf{count}, \mathsf{cdata}, [\mathsf{tags}], \sigma)$ *at time* atime*, an entry is added to the ledger. Without loss of generality we can trivially extend this algorithm to take as input a list of activities, each generating individual entries in the ledger.*

- IProof *is an interactive protocol between a user* $(\mathsf{sk}, \mathsf{pk})$ *and a verifier to check whether the user satisfies the policy* Ψ*. Both parties have access to* PL*, and can easily search and extract information from the public ledger.*
  - $\mathsf{IProofU}(\mathsf{pp}, \mathsf{sk}, \Psi)$ *is an algorithm run by the user that interacts with* IProofV *to authenticate the user and prove its compliance with the policy* Ψ*. The algorithm completes successfully with* succ *or aborts with* abort*.*
  - $\mathsf{IProofV}(\mathsf{pp}, \mathsf{pk}, \Psi)$ *is an algorithm run by a verifier to assess if the user with* pk *satisfies* Ψ*. It searches through the ledger, by calling* $\mathsf{PL.Search}(Q)$*, for a search query* Q *built from* Ψ *for user* pk*. Using the obtained list of valid entries, it interacts with* IProofU *to ensure the user proves that he satisfies the policy* Ψ*. The output of this algorithm is a boolean value that is set to true if the user can be authenticated and he satisfies the policy; and false otherwise.*

The system satisfies the following correctness property:

$$\langle \mathsf{succ}, \mathsf{true} \rangle \leftarrow \langle \mathsf{IProofU}(\mathsf{pp}, (\mathsf{pk}, \mathsf{sk}), \Psi), \mathsf{IProofV}(\mathsf{pp}, \mathsf{pk}, \Psi) \rangle ,$$

that holds for any policy Ψ, any $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$, and any user $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGenU}(\mathsf{pp})$, if there exists A such that $\Psi(A) = 1$ and :

$$A = \left\{ \begin{array}{l|l} & \text{user } (\mathsf{sk}, \mathsf{pk}) \text{ has added entry} \\ (\mathsf{atime}, \mathsf{aname}, \mathsf{count}, & \quad (\mathsf{pk}, \mathsf{atime}, \mathsf{aname}, \mathsf{count}, \mathsf{cdata}, [\mathsf{tags}], \sigma) \\ \;\mathsf{adata}, [\mathsf{tags}]) & \text{to PL by making the call} \\ & \quad \mathsf{SubmitU}(\mathsf{pp}, \mathsf{sk}, (\mathsf{aname}, \mathsf{adata}, [\mathsf{tags}])) \end{array} \right\} .$$

Remark 4 (TAP with external database). We can extend Definition 1 to account for the removal of the activity data by the user. For this we can introduce the algorithm Remove that calls $\mathsf{PL.Remove}(\mathsf{pk}, \mathsf{aname}, \mathsf{count}, \mathsf{cdata}, [\mathsf{tags}], \sigma)$ defined in Section 2.1. Since the commitments to all records remain in the public ledger, users can also temporary remove their activity records from the external database if they do not want to delete them completely.

2.5 Security Properties

In this section we introduce the security properties TAP schemes should satisfy. They are centered around the confidentiality of activity data, and the policy verification of authenticated users.

Oracles for $\mathcal{A}$*.* The adversary $\mathcal{A}$ has access to the list of entries in the public ledger PL and can interact with it to add or search for entries. Furthermore, he has access to the functionalities of TAP and can submit activities in the name of some honest user, for

```
$Exp^{\mathsf{dc},\beta}_{\mathcal{A},\mathsf{TAP}}(\lambda)$
$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGenU}(\mathsf{pp})$
$(\mathsf{aname}, \mathsf{adata}_1, \mathsf{adata}_2, [\mathsf{tags}]) \leftarrow \mathcal{A}_1^{\mathsf{Append}(\cdot),\mathsf{Search}(\cdot),\mathsf{Submit}(\cdot),\mathsf{IProofU}(\cdot)}(\mathsf{pp}, \mathsf{pk})$
$(\mathsf{pk}, \mathsf{atime}, \mathsf{aname}, \mathsf{count}, \mathsf{cdata}_\beta, [\mathsf{tags}], \sigma_\beta) \leftarrow$
    $\mathsf{SubmitU}(\mathsf{pp}, \mathsf{sk}, (\mathsf{aname}, \mathsf{adata}_\beta, [\mathsf{tags}]))$
$\beta' \leftarrow \mathcal{A}_2^{\mathsf{Append}(\cdot),\mathsf{Search}(\cdot),\mathsf{Submit}(\cdot),\mathsf{IProofU}(\cdot)}(\mathsf{atime}, \mathsf{count}, \mathsf{cdata}_\beta, \sigma_\beta)$
return $\beta' \wedge \mathcal{A}$ did not call $\mathsf{IProofU}(\mathsf{pp}, \mathsf{sk}, \Psi)$ with
  $(\mathsf{pk}, \mathsf{atime}, \mathsf{aname}, \mathsf{count}, \mathsf{cdata}_\beta, \sigma_\beta, [\mathsf{tags}]) \in \mathsf{Search}(Q)$
  for $Q$ - search query based on $\Psi$ and $\mathsf{pk}$
```

Fig. 1: Data confidentiality experiment

which he does not know the secret key, and play the role of any of the participants in the interactive proof protocol. All oracles are based on the algorithms in PL and TAP:

- $\mathsf{Append}(\cdot)$: It calls $\mathsf{PL.Append}(\cdot)$, and returns its output;
- $\mathsf{Search}(Q)$: It calls $\mathsf{PL.Search}(Q)$, and returns its output;
- $\mathsf{Cor}(\mathsf{pk})$: It returns the sk correspond to the pk used in the two experiments;
- $\mathsf{Submit}(\cdot, \cdot, [\cdot])$: It returns the output of $\mathsf{TAP.SubmitU}(\mathsf{pp}, \mathsf{sk}, (\cdot, \cdot, [\cdot]))$, for the user $(\mathsf{sk}, \mathsf{pk})$ generated in the main body of the two experiments;
- $\mathsf{IProofU}(\cdot)$: The adversary plays the role of a malicious verifier in the interactive IProof protocol with an honest user - that calls $\mathsf{TAP.IProofU}(\mathsf{pp}, \mathsf{sk}, \cdot)$;
- $\mathsf{IProofV}(\cdot)$: The adversary plays the role of a malicious prover in the interactive IProof protocol with an honest verifier - that calls algorithm $\mathsf{TAP.IProofV}(\mathsf{pp}, \mathsf{pk}, \cdot)$.

Data Confidentiality. A fundamental property of TAP is the *privacy of data* that ensures no information concerning the data of an activity is leaked by the entry in the ledger for that activity, except the information provided from the name, counter and tags assigned to it. This is formalized in Fig. 1, where we consider a PPT adversary $\mathcal{A}$ that is required to distinguish between two private data values by seeing an entry in the ledger of one of them. The activity data that is transformed in an entry, is independent of the adversary and we parametrize the experiment with a random bit β to mark this. Moreover, this entry is attributed to a user for which the adversary doesn't know the secret key. The adversary can use oracle access to submit activities in the name of this user, or ask for proofs for policies that do not contain the entry he is challenge on. This restriction is needed to remove trivial attacks where the adversary can win the game by asking for a policy that checks if there exists an entry that contains one of the private data. In addition to entries made for this user pk, the adversary can introduce in the ledger entries for other users which he controls (knows their secret key).

Definition 2 (Data Confidentiality). *A* TAP *scheme satisfies data confidentiality, if no PPT adversary* $\mathcal{A}$ *can distinguish between* $Exp^{\mathsf{dc},0}_{\mathcal{A},\mathsf{TAP}}$ *and* $Exp^{\mathsf{dc},1}_{\mathcal{A},\mathsf{TAP}}$ *defined in Fig. 1,*

```
$Exp^{\mathsf{apc}}_{\mathcal{A},\mathsf{TAP}}(\lambda)$
$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$
$(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KGenU}(\mathsf{pp})$
$\mathcal{A}^{\mathsf{Append}(\cdot),\mathsf{Search}(\cdot),\mathsf{Cor}(\cdot),\mathsf{Submit}(\cdot),\mathsf{IProofU}(\cdot),\mathsf{IProofV}(\cdot)}(\mathsf{pp},\mathsf{pk})$
return $\mathcal{A}$ made a call to $\mathsf{IProofV}(\cdot)$ with
  $\mathsf{true} \leftarrow \mathsf{IProofV}(\mathsf{pp},\mathsf{pk},\Psi) \wedge \neg\Psi(\emptyset) \wedge \mathcal{A}$ not running $\mathsf{IProofU}(\cdot) \wedge$
  ( $\mathcal{A}$ did not call $\mathsf{Cor}(\mathsf{pk}) \vee$
    $\forall\, e = (\mathsf{pk},\cdot,\cdot,\cdot,\cdot,[\cdot]) \in \mathsf{Search}(Q)$, for $Q$ based on $\Psi$ and $\mathsf{pk}$,
        $\mathcal{A}$ did not not add $e$ using $\mathsf{Append}(\cdot) \wedge$
    $\neg\Psi(A)$ for $A = \{(\mathsf{pk},\mathsf{atime},\mathsf{aname},\mathsf{count},\mathsf{adata},[\mathsf{tags}])|$ for
        entries added by $\mathsf{Submit}(\cdot)$ to the ledger $\mathsf{PL}\}$ )
```

Fig. 2: Authenticated Policy Compliance experiment

i.e., the following advantage is negligible in λ*:*

$$\mathsf{Adv}^{\mathsf{dc}}_{\mathcal{A},\mathsf{TAP}} = \left|\Pr\left[Exp^{\mathsf{dc},0}_{\mathcal{A},\mathsf{TAP}}(\lambda) = 1\right] - \Pr\left[Exp^{\mathsf{dc},1}_{\mathcal{A},\mathsf{TAP}}(\lambda) = 1\right]\right|.$$

Authenticated Policy Compliance. This property ensures only authorized users can prove to an honest verifier they satisfy the verifier's policy. It requires the adversary $\mathcal{A}$ to "forge" this authorization by either *impersonating a user* that may satisfy the policy and for which the adversary doesn't know the secret key, or by *faking the evidence* for activities he did not put into the ledger (even if he knows the secret key). To evaluate if the adversary is capable of providing fake evidence when he does not satisfy the policy, we need to apply the policy over all activities that have been submitted for a single user. As the activity data is relevant in establishing if a policy is satisfied, we restrict the adversary to provide evidence only for policies that consider activities added by Submit. Furthermore, the policy that is verified must be valid, $\neg\Psi(\emptyset)$, that is it must not return true independent of the input. We formalize these conditions in Fig. 2, where we consider a PPT adversary that submits entries to the ledger and wins if he can convince an honest verifier to return true. Because of the interactive nature of the algorithm in IProof we restrict the adversary to not call the verification oracle $\mathsf{IProofV}$ in the same time an instance of $\mathsf{IProofU}$ is opened.

Definition 3 (Authenticated Policy Compliance). *A* TAP *scheme ensures authenticated policy compliance, if no PPT adversary* $\mathcal{A}$ *can win the experiment* $Exp^{\mathsf{apc}}_{\mathcal{A},\mathsf{TAP}}$ *defined in Fig. 2, i.e., the following advantage is negligible in* λ*:*

$$\mathsf{Adv}^{\mathsf{apc}}_{\mathcal{A},\mathsf{TAP}} = \Pr\left[Exp^{\mathsf{apc}}_{\mathcal{A},\mathsf{TAP}}(\lambda) = 1\right].$$

3 Our TAP construction

In this section we describe our general construction and analyze its security properties. We start with standard cryptographic primitives [10] that are used as building blocks, and the method to compute encryption keys that hide the activity data.

3.1 Building Blocks

Our construction relies on a *pseudo-random function* $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\star \rightarrow \{0,1\}^{poly(\lambda)}$ [8], and an *existentially unforgeable digital signature scheme* $\mathsf{DS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ [6]. We rely further on a *symmetric encryption scheme* $\mathsf{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ with two security requirements: *indistinguishability under chosen plaintext attack (IND-CPA)*, and *wrong-key detection (WKD)* [2]. The WKD property is not standard and has been introduced to bound the winning probability of an adversary in successfully decrypting a ciphertext with two different keys. More precisely, any efficient adversary has a negligible probability to win the game where he computes different keys $k \neq k'$ s. t. the expression $\mathsf{Dec}(k', \mathsf{Enc}(k, m)) \neq \perp$ holds, for all messages m. The WKD property is used in Lemma 2, see also Remark 6.

3.2 Key Management

Activities are submitted directly by users who control the type of information they add to the ledger. For each activity $(\mathsf{pk}, \mathsf{atime}, \mathsf{aname}, \mathsf{count}, \mathsf{adata}, [\mathsf{tags}])$, the activity data adata is private, and is never stored in plain. This component in the activity is encrypted with a symmetric encryption scheme SE before being appended to the ledger. The encryption keys are an important part of the policy verification proof, and as such the user is required to store them securely.

Our solution for key management is to use the PRF to derive the encryption keys on demand, and reduce the amount of storage space on the user's side. For each user, a random seed $s \in \{0,1\}^\lambda$ is chosen, and *time interval keys* $tk_i = \mathsf{PRF}_s(\mathsf{pk}, i)$ are derived for the time interval i first. These are then used to derive *activity type keys* $ak_j = \mathsf{PRF}_s(tk_i, j)$ for some activity name aname_j in time interval i. Finally, the *activity record keys* $rk_k = \mathsf{PRF}_s(ak_j, \mathsf{count}_k)$ are derived and used by the user to encrypt the activity data. During the submission process of an activity (algorithm $\mathsf{SubmitU}$ in Fig. 3), first the time interval is defined by $i \leftarrow \mathsf{PL.GetInterval}(\mathsf{atime}_i)$, followed by the computation for the record key

$$rk_k = \mathsf{PRF}_s(\mathsf{PRF}_s(\mathsf{PRF}_s(\mathsf{pk}, i), \mathsf{aname}_j), \mathsf{count}_k),$$

given the activity $(\mathsf{pk}, \mathsf{atime}_i, \mathsf{aname}_j, \mathsf{count}_k, \mathsf{adata}, [\mathsf{tags}])$.

An interesting and useful feature of this key management approach is that all activity record keys are overwhelmingly unique. This is ensured by the pseudo-randomness of the PRF, and the uniqueness of the inputs.

3.3 Generic Construction

In the following we provide a high-level intuition behind our construction, which is specified in Fig. 3 and 4. A trusted third-party initializes the public ledger PL and cre-

```
Setup($\lambda$)
1 :  $D \leftarrow$ define ANames
         and TNames
2 :  pp $\leftarrow$ PL.Setup($\lambda$)
3 :  return (pp, $D$)

KGenU(pp)
1 :  $s \leftarrow_\$ \{0,1\}^\lambda$
2 :  (sigk, pk) $\leftarrow$ DS.KGen($\lambda$)
3 :  return ((sigk, $s$), pk)

SubmitU(pp, (sigk, $s$), (aname, adata, [tags]))
1 :  atime $\leftarrow$ PL.GetTime()
2 :  $i \leftarrow$ PL.GetInterval(atime)
3 :  $L \leftarrow$ PL.Search("all (pk, atime, aname, $\cdot$, $\cdot$, [$\cdot$])
         with $i \leftarrow$ PL.GetInterval(atime)$''$)
4 :  if ($L = \emptyset$) then count $\leftarrow 0$
5 :    else count $\leftarrow$ last counter from L + 1
6 :  $rk \leftarrow$ PRF$_s$(PRF$_s$(PRF$_s$(pk, $i$), aname), count)
7 :  cdata $\leftarrow$ SE.Enc($rk$, adata)
8 :  $\sigma \leftarrow$ DS.Sign(sigk, (aname, count, cdata, [tags]))
9 :  PL.Append(pk, aname, count, cdata, [tags], $\sigma$)
```

Fig. 3: The setup, user key generation, and submission of activities for TAP.

ates public parameters which includes the initialization of the append-only list, and the clock. Moreover, he defines the set of allowed activity names and tags.

Users generate their own signing/verification key pair (sigk, pk) for the digital signature scheme DS. Additionally, they generate a seed s that is used to derive a time-dependent key hierarchy for submitting activities. The pk is used as public identifier for the user in the system, while sk $=$ (sigk, s) should be stored securely. The user may have multiple identities in our system, by generating new signing keys and a new seed.

Activities (aname, adata, tags) are submitted directly by users who control what type of information is added to the ledger. Each data adata is stored encrypted with a unique key that follows the procedure described in the Section 3.2. To authenticate the activity, the user provides a signature before appending it to the ledger.

The verification of historic evidence is performed interactively between a user (sk, pk) with sk $=$ (sigk, s) and a verifier w.r.t the policy Ψ. We formalize this interaction in Fig. 4, where both parties communicate over an authenticated and confidential channel. Both the user and the verifier have access to the public ledger PL. The verifier searches PL for entries that match the descriptors from the policy for this user, and returns false if any of them are invalid. Then, it sends to the user the list L obtained from the search with a fresh nonce c. The user checks if the list of entries he received is identical to what he has retrieved from the ledger, and aborts if they do not match. The user re-computes the activity record keys for the records in L, and gives them as part of the key set K to the verifier together with a signature over the nonce received. The verifier returns true, if upon successful decryption using the received keys the activities satisfy Ψ, and the signature verifies.

Remark 5 (Correctness). The correctness of TAP is ensured by the method in which activity record keys are computed. The keys derived upon submission of activity data in Line 6 of SubmitU, in Fig. 3, are identical with the activity record keys computed by the user in Fig. 4. Because the PL.Append in Line 9 of SubmitU in Fig. 3 calls internally atime$'$ $\leftarrow$ GetTime(), it is important that atime and atime$'$ map to the same interval i.

```
$\mathsf{IProofU}(\mathsf{pp}, \mathsf{sk}, \Psi)$ with $\mathsf{sk} = (\mathsf{sigk}, s)$          $\mathsf{IProofV}(\mathsf{pp}, \mathsf{pk}, \Psi)$

$Q \leftarrow$ construct from $\Psi$ and pk                        $Q \leftarrow$ construct from $\Psi$ and pk
                                                                   $L \leftarrow \mathsf{PL.Search}(Q)$
                                                                   $c \leftarrow\$ \{0,1\}^{poly(\lambda)}$
                                                                   for $(\mathsf{pk}, \mathsf{atime}, \mathsf{aname}, \mathsf{count}, \mathsf{cdata}, [\mathsf{tags}], \sigma) \in L$
                                                                     if $\neg\mathsf{DS.Verify}(\mathsf{pk}, (\mathsf{aname}, \mathsf{count}, \mathsf{cdata}, [\mathsf{tags}]), \sigma)$
                                                                      then return false
                                   $\xleftarrow{c, L}$
if $(L \neq \mathsf{PL.Search}(Q))$ then return abort
$K \leftarrow \emptyset$                                           $A \leftarrow \emptyset$
for $(\mathsf{atime}, \mathsf{aname}, \mathsf{count}, \mathsf{cdata}, [\mathsf{tags}]) \in L$ do
  $i \leftarrow \mathsf{PL.GetInterval}(\mathsf{atime})$
  $rk \leftarrow \mathsf{PRF}_s(\mathsf{PRF}_s(\mathsf{PRF}_s(\mathsf{pk}, i), \mathsf{aname}), \mathsf{count})$
  $K \leftarrow K \cup \{rk\}$
$\sigma_P \leftarrow \mathsf{DS.Sign}(\mathsf{sigk}, (c, K))$     $\xrightarrow{S, \sigma_P}$
                                                                   for $0 \leq i \leq |L|$ do
                                                                     $\mathsf{data} \leftarrow \mathsf{SE.Dec}(K[i], L[i])$
                                                                     $A \leftarrow A \cup \{\mathsf{atime}, \mathsf{aname}, \mathsf{data}, [\mathsf{tags}]\}$
return succ                                                        return $\Psi(A) \wedge \mathsf{DS.Verify}(\mathsf{pk}, (c, K), \sigma_P)$
```

Fig. 4: The interactive protocol IProof for TAP. It is assumed that the user and verifier are communicating over an authenticated and confidential channel.

3.4 Security Analysis

Lemma 1. *The* TAP *construction in Fig. 3 and 4 offers data confidentiality if* SE *is IND-CPA and* PRF *is pseudo-random.*

Proof. All ciphertexts in the ledger are encrypted with unique keys derived via PRF using unique public key-seed (pk, s) pair and a unique triple composed of the activity name, time frame, and counter. Using the pseudo-random property of PRF and the fact that s is kept secret from the adversary, we can be sure the adversary obtains only a negligible advantage, by just looking at entries in the ledger for this user pk.

Our construction provides this unique key per activity as proof in the interactive IProof protocol, when the adversary plays the role of a verifier and calls the oracle $\mathsf{IProofU}(\mathsf{pp}, \mathsf{sk}, \cdot, \cdot, [\cdot])$. First, we show that the adversary has negligible advantage to obtain the key used for the challenge ciphertext - that encrypts adata_β.

Before the challenge, the adversary may call the IProofU oracle and ask for keys of (valid) entries in the ledger. Even if the adversary can predict what input the PRF will take, i.e., name, time and counter for an activity, $\mathcal{A}_1$ obtains only negligible information due to the pseudo-randomness property of PRF. The component $\mathcal{A}_2$ is restricted to not ask this exact key from the IProofU queries. Moreover, all subsequent proofs - keys, that may be linked to the key used to encrypt the challenge also produce a negligible advantage via the pseudo-randomness property of PRF. Therefore, the adversary only obtains a negligible advantage w.r.t the encryption key for the challenge.

We conclude this proof by using the IND-CPA security of the symmetric encryption scheme SE that encrypted the challenge, as the adversary doesn't have access to the encryption/decryption key.

Lemma 2. *The* TAP *construction in Fig. 3 and 4 ensures authenticated policy compliance, if* DS *is existentially unforgeable,* SE *is WKD, and* PRF *is pseudo-random.*

Proof. We upper bound the advantage of the adversary in winning this game by the following two probabilities, with all of them restricting the adversary not to run $\mathsf{IProofU}$ when running the particular instance of $\mathsf{IProofV}$ used in the experiment outcome:

- Exp_1 : the adversary *convinces a honest verifier to return true when he doesn't ask for the secret key*. This is defined exactly as experiment in Fig. 2 where we remove the second part of the disjunction in the return.
- Exp_2 : the adversary *convinces a honest verifier to return true when he doesn't satisfy the policy*. This is defined exactly as experiment in Fig. 2 where we remove the first part of the disjunction in the return.

Bound on Exp_1. The secret key of a user contains a signing key sigk and a seed s. We use the fact the adversary does not know sigk, and can not provide real signed messages with this key when impersonating the user. To convince the verifier to return true (on any adversarial policy) the adversary needs to sign the message that gives the encryption keys. This message is a new message in the system for which there should exists no signature, as it contains a fresh commitment given by $\mathsf{IProofV}$. Therefore, the adversary must forge a signature and break the existential unforgeability of DS. To complete that reduction to the security of DS we are using the fact the adversary can not run a man-in-the-middle attack where he uses $\mathsf{IProofU}$ to create proofs for this verifier.

Bound on Exp_2. The adversary to “show” he satisfies a policy when in fact he doesn't, must claim that an entry corresponds to an activity different than the one he has actually submitted. This claim is with respect to the data in the activity, otherwise he would have directly satisfied the policy if the data was not required. More precisely, the adversary needs to provide a decryption key such that he obtains a valid decryption different from the one submitted. The WKD property of SE ensures that if the adversary uses a decryption key different from the one used to encrypt, then the ciphertext would not decrypt to anything valid. As such the adversary would need to compute a decryption key that coincides with the encryption key, and this is deemed improbable by the pseudo-randomness property of PRF.

Remark 6. While the wrong-key detection property is required for the proof of Lemma 2, we observe that in practice this requirement can be lifted. This is because, for current practical symmetric encryption schemes, e.g. AES based, if a different key is used, then the message would look random, and make it unlikely to obtain *something meaningful*. Therefore, the policy would not succeed given such random-looking messages.

4 Conclusion and Future Directions

Timeline Activity Proofs (TAP) proposed in this work allow users to store their online interactions and build a timeline of their activities. These can later be used by users to prove statements that consider past activities and by this increase the trustworthiness of their online identities (e.g. profiles). Additionally, we propose a construction that gives access to plain data to verifiers capturing today's method for social login. As a future research direction it would be interesting to see if TAP can be extended with functionalities that allow for certain statements to be proven without disclosing the decryption keys. This brings us closer to the zero-knowledge techniques used in *(decentralized) anonymous credentials* [3, 7].

Another possible research direction is to consider a feedback mechanism. Currently, users self-issue activities and trust is ensured by the amount of information the user is providing. Using a feedback mechanism to rate these activities, e.g., the one used by Amazon, eBay to rate sellers, could increase the trust guarantees further.

Acknowledgements. Constantin Cătălin Drăgan and Mark Manulis were supported by the EPSRC project TAPESTRY (EP/N02799X).

References

1. A. Barth, D. Boneh, and B. Waters. Privacy in Encrypted Content Distribution Using Private Broadcast Encryption. In *Financial Cryptography and Data Security*, pages 52–64, 2006.
2. R. Canetti, Y. T. Kalai, M. Varia, and D. Wichs. On Symmetric Encryption and Point Obfuscation. In *TCC'10*, pages 52–71, 2010. LNCS 5978.
3. D. Chaum. Security Without Identification: Transaction Systems to Make Big Brother Obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.
4. R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACMCCS*, pages 79–88, 2006.
5. L. A. Cutillo, R. Molva, and T. Strufe. Safebook: A privacy-preserving online social network leveraging on real-life trust. *IEEE Communications Magazine*, 47(12):94–101, 2009.
6. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.
7. C. Garman, M. Green, and I. Miers. Decentralized Anonymous Credentials. In *NDSS*, 2014.
8. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.
9. F. Günther, M. Manulis, and T. Strufe. Cryptographic Treatment of Private User Profiles. In *Financial Cryptography Workshops'11*, pages 40–54, 2011.
10. J. Katz and Y. Lindell. *Introduction to Modern Cryptography, 2nd Edition*. CRC Press, 2014.
11. N. C. Matthew Rosenberg and C. Cadwalladr. How Trump Consultants Exploited the Facebook Data of Millions. 2018/03/17. `https://www.nytimes.com/2018/03/17/us/politics/cambridge-analytica-trump-campaign.html`.
12. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
13. C. Nguyen. China might use data to create a score for each citizen based on how trustworthy they are. 2016/10/26. `http://uk.businessinsider.com/china-social-credit-score-like-black-mirror-2016-10`.
14. OpenID Connect Framework. `https://openid.net`, Accessed at 2018/06/18.
15. Smartbit. `https://www.smartbit.com.au`, Accessed at 2018/06/18.
16. I. Symeonidis, P. Tsormpatzoudi, and B. Preneel. Collateral damage of Facebook Apps: an enhanced privacy scoring model. IACR Cryptology ePrint Archive, Report 2015/456.
17. N. Tiku. Facebook will make it easier for you to control your personal data. 2018/03/28. `https://www.wired.com/story/new-facebook-privacy-settings`.
18. G. Wood. Ethereum yellow paper, 2014.