Least Authority
PRIVACY MATTERS

ZKAPs Whitepaper

Published: July 2021

What are ZKAPs?

- Zero-Knowledge Access Passes: Anonymous authorization tokens
- Utilized as a protocol to enable users to access services without revealing personal information
- A modified version of Brave's implementation of Privacy Pass
- Adapted for PrivateStorage, a distributed and anonymous file storage system

Who is this paper for?

This paper is intended to be accessible to audiences without any previous knowledge of cryptography or programming. Nevertheless, it provides ample detail and references for those who are interested in the cryptographic and implementation details of ZKAPs.

Traditional, account-based authorization has privacy issues: all actions or data of a user can be collated to form a profile, which is often linked to the user's real-life identity as well. ZKAPs provide a way to decouple a user's individual actions and data from each other and from the user's identity, thereby increasing anonymity (section 1).

Section 2 describes the process of issuing and redeeming ZKAPs. In the issuance phase, the user's client generates a batch of random tokens, blinds them, and sends them to the issuer. The issuer signs the blinded tokens without knowing the token numbers or the user's identity. In the redemption phase, the user unblinds the signed tokens and redeems them in exchange for some service. Because token numbers are random, the unblinded signed tokens contain no information about the user or about other tokens in the same batch.

ZKAPs are a modified version of Privacy Pass, which has been developed as way to help users avoid having to solve too many CAPTCHAs. By solving one challenge, Privacy Pass users get a batch of anonymous tokens which they can use to get access to online content. Brave has developed a Rust implementation of Privacy Pass, which is the basis of ZKAPs. Instead of proof of humanness for Content Delivery Networks, ZKAPs serve as proof of payment to be redeemed for storage time in Least Authority's anonymous file storage system, PrivateStorage (section 3).

While ZKAPs provide a privacy-enhancing alternative to account-based identification, they do not guarantee complete anonymity: users' privacy can still be compromised by metadata and the low number of total users of a service. Some further limitations of the current implementation of ZKAPs are also discussed in section 4.

Currently, ZKAPs are used as proof of payment to be redeemed for file storage in PrivateStorage. However, they can be used as proof of other attributes, for example, as proof of age, copyright or membership in an organization. Potential use cases for ZKAPs, including public transportation systems, VPN, direct messaging and Content Delivery Networks, are described in section 5.

Table of Contents

1

Why do we need ZKAPs?

Problems with account-based identification:

- User accounts for online services allow the provider to track and connect that user's behaviour and their data, creating user profiles
- Connecting user accounts to real-life identities (e.g., via payment processing) compromises the privacy of users' actions and data

Privacy-enhanced identification using ZKAPs:

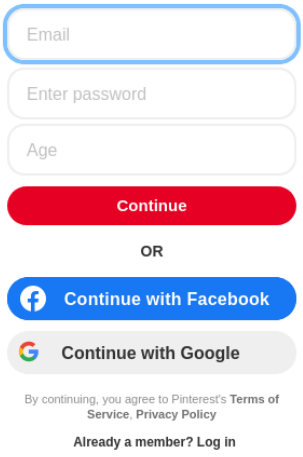
- Individual user actions and pieces of data are not linked, so they do not form a user profile
- Payment data are separate from usage data, so they do not reveal the real-life indentity of users to the service provider

1.1 The problems with user accounts

Using an online service requires users to prove that they have authorization to access the service. Most often, the right to use the service is granted after registration, and users have to identify themselves to gain access.

For paid services, each user account is typically connected to the payment processor in order to verify if they have paid the fees required to access the digital asset.

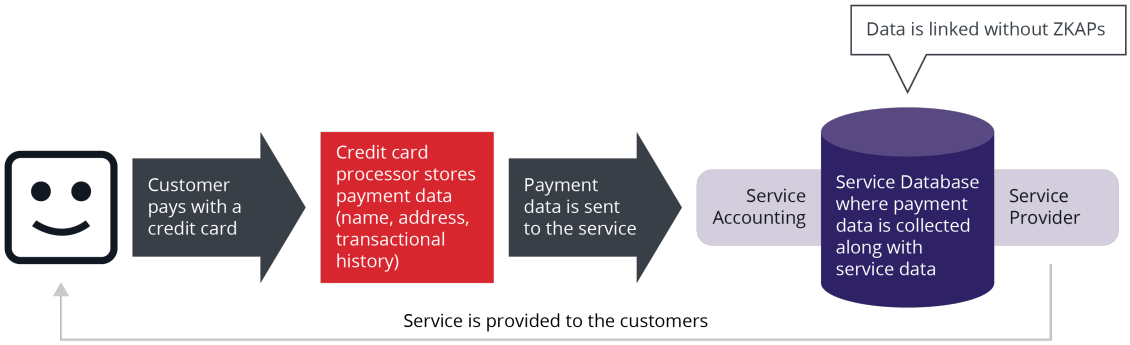
Such an account is a "durable identifier", meaning that the same email address or username is connected to all instances of the user interacting with the service.



In order to process payments, service providers collect users' personal data:

- Name
- Email address
- Location (for VAT)
- Transaction data
- Credit/Debit card number

These data are stored by the service providers, and can be used in ways the user did not consent to and not necessary to operate the service: providers can construct profiles of users' purchases, and use these for targeted advertisement, individualized pricing, or forward/sell it to third parties.

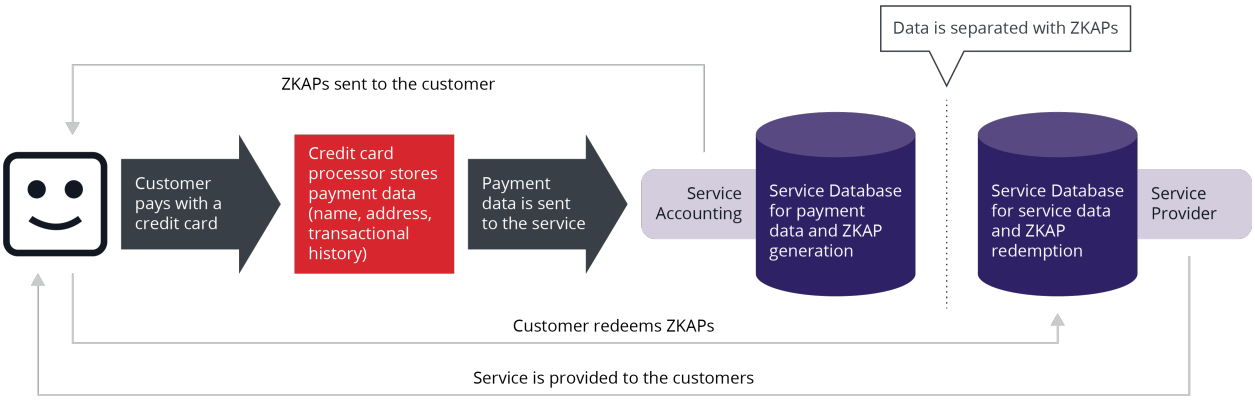


On the other side, handling personal data creates a lot of extra work for service providers because of the complicated and geographically different data handling regulations. Not all providers can manage this task in a sufficiently secure way, and if the account becomes compromised by someone stealing or guessing the password, the attacker has full control of the account and knowledge of the user's past actions and personal data. Apart from negatively affecting users, such scenarios pose serious risks to service providers as well, as discussed in section 5.

1.2 An alternative to user-based verification

Most service providers do not need personal data to provide the service, they only need it to process the payment ("accounting").

ZKAPs provide a way to keep payment data separate from personal data. Rather than linking the payment database to the service database, users are given ZKAPs in exchange for proving that they have permission to use the service. This can be a proof of payment, proof of age, proof of membership in an organization, etc. Users can then redeem these tokens to anonymously prove they have permission to use a certain service. With this model, the service provider knows if the user is authorized to use the service, but not who the user is.



In other words, ZKAPs are used as an **anonymous proof of payment**.

1.3 Towards anonymous authorization

Authorization based on user accounts is similar to a guest list for a party: to gain access, guests need to present proof of their identity, like an ID card or driver's license. The person at the door checks the name on the guest's ID to see if that name is on the guest list, and they also check their picture to see if the ID really belongs to the person trying to gain access.
Similarly, to access a service with account-based identification, the service provider checks if a user's account has permission to access the service, and users also need to provide their unique user name and password to verify their identity.

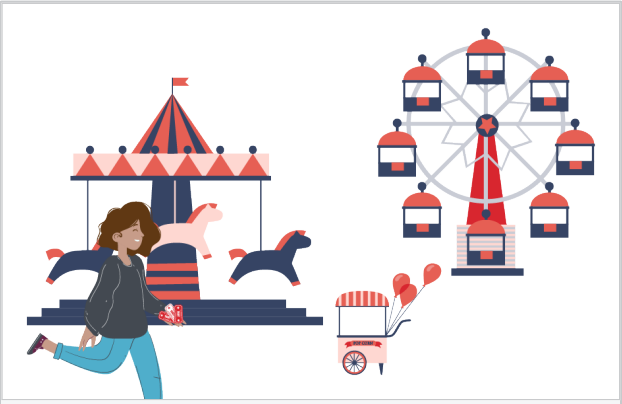
Conversely, an anonymous authorization protocol does **not** require users to reveal their identity to access the service; they are only required to prove that they have permission to use the service.

A rudimentary form of anonymous authorization is a ticket roll: customers buy tickets containing numbered labels, and then use them to access services or buy items. The numbers on the ticket help merchants verify the validity of the tickets. For example, they can look at whether the number on the ticket falls within some particular, expected range and grant or deny access to the customer based on that.

Each ticket has a corresponding coupon (as seen here on the bottom of the ticket roll), which has the same number as the ticket. The customer keeps the coupon, which helps with record-keeping and serves as a means of dispute-resolution. For example, if a merchant doubts the authenticity of a given ticket, the customer can prove it by presenting the corresponding coupon.

Let's see how this works in practice. Our protagonist, Aditi, has heard that a carnival has come to town and decides to check it out.

At the carnival, Aditi purchases a roll of tickets to be exchanged for rides, games, shows food, and so on.



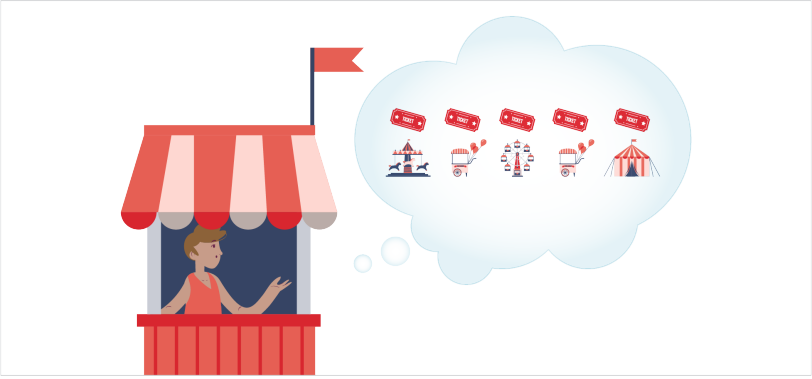
She goes for a spin on the merry-go-round, has some popcorn, and takes a ride on the ferris wheel.

Then she gets some more popcorn and catches the circus act at the big tent.

After a day of fun, Aditi finally goes home satisfied.

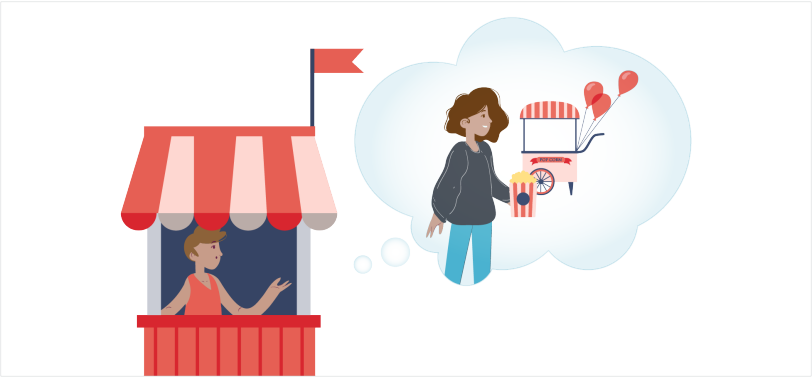
In some respects, the ticket system appears to be a privacy-improvement: Aditi didn't have to show her ID at each ride and authenticate herself: showing the ticket is proof enough that she paid and is allowed to go on the rides and get snacks.

Nevertheless, this ticket system is not completely private: the properties of the tickets themselves can easily be exploited for privacy violations. Imagine a situation in which a malicious ticket-booth operator – Mallory – wants to figure out what Aditi did while at the carnival.



Tickets that are collected after the fact – and, more specifically, the range and order of numbers on those tickets – can be used to deduce certain patterns of behavior – for example, that the same person bought some popcorn, rode the ferris wheel, took a spin on the merry-go-round, ate some *more* popcorn, saw a show in the big tent and went home.

Furthermore, the existence of the coupon reel retained by Mallory at the ticket booth can be used to determine which person purchased which range of numbered tickets; she could easily staple Aditi's tickets to her merchant copy of Aditi's receipt.



Taken together, the basic properties of the authorization protocol itself can thus be used to determine what Aditi did at the carnival – and, in this case, that she really loves popcorn.
Maybe Mallory could sell this information to popcorn vendors – or use the information to lure Aditi into other behaviors.

So, while tickets (as a broad authorization system) might represent a privacy improvement over user accounts (an authentication system), a sufficiently-equipped attacker can still use the information leaked by them to monitor, track, or surveil who use it – and in certain respects, this is precisely the point.

Of course, this does not mean that we should do away with authorization protocols entirely and have only, say, "permissionless" carnivals – a carnival operator who operates a "permissionless" carnival, after all, would surely go out of business fast! But the question still remains: how can we accept legacy payments with built in tracking mechanisms (such as credit cards) without jeopardizing the privacy of users?

2

How ZKAPs work

There are two phases of using ZKAPs:

1. Issuance phase

A customer pays for some passes, generates a batch of randomly numbered tokens, blinds them, and sends them to the issuer along with the proof of payment. The issuer signs the blinded tokens without knowing the customer's identity or the token numbers, and sends these back to the customer.

2. Redemption phase

To "spend" the tokens, the customer unblinds them, and sends them to a provider in exchange for some service. The service provider checks the signature and the token number for validity, but they do not know which tokens belong to which customer, nor which tokens belong to the same batch.

ZKAPs attempt to solve the problem of privacy-conscious authorization: they can be used to authorize individual actions without identifying individual users.

At a higher level, ZKAPs are like carnival tickets without consecutive numbering or any other tracking information – the customer spends or uses them like a currency, while denying the service provider who accepts them the ability to stitch together a behavioral profile about the customer after the fact.

This approach to ZKAPs is called **Privacy by Design**: the service provider cannot uncover the user's identity even if they want to. This is in contrast with **Privacy by Policy**, where the service provider has access to users' data, which is only limited by laws or policies.

Privacy by Policy

- Service provider has access to users' personal data
- Can connect a specific user's actions, data and/or behaviour to the user's real world identity
- Only laws/policies in the way of misusing data
- Possibility of data breach via hacking or force (e.g. governments, intelligence agencies)

Privacy by Design

- Service provider **does not** have access to users' personal data
- **Cannot connect** a specific user's actions, data and/or behaviour to the user's real world identity
- Personal information cannot be leaked or stolen – the service provider does not have it
- A specific user's actions/data/behaviour **cannot** be handed over to third parties (including government agencies) – the service provider does not have these

It is important to note that Privacy by Design is not an absolute, and it does not apply equally to all properties of a given system. Some caveats of the privacy properties of ZKAPs are discussed in 4.

2.1 A short summary of the process of using ZKAPs

ZKAPs are created by the interaction of two parties:

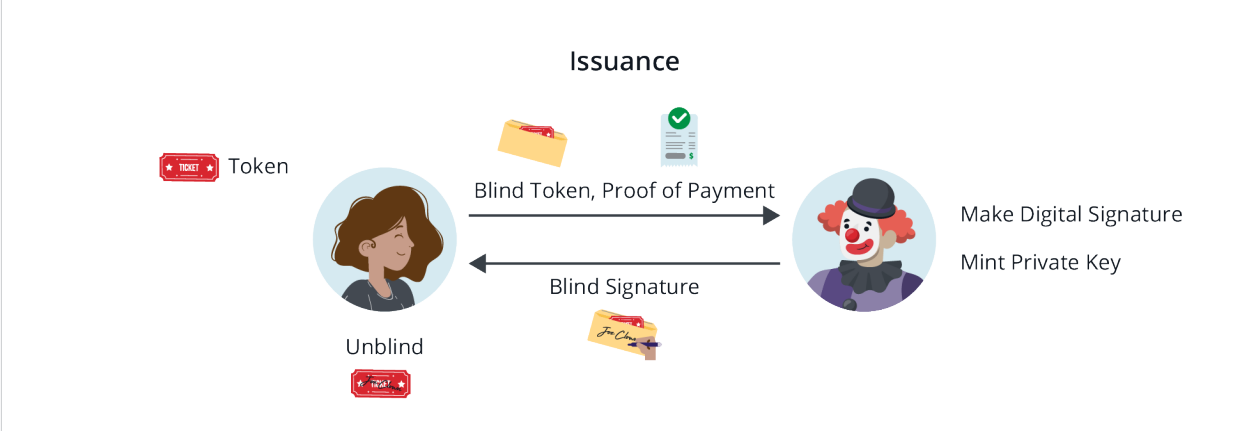
1. the **user**: the customer purchasing the ZKAPs, typically by using an app or an extension on their device.
2. the **issuer**: the entity issuing ZKAPs, which is also the entity providing some service (e.g. data storage or content) in exchange for ZKAPs.

Authorization with ZKAPs happens in 2 phases: the issuance phase and the redemption phase. These are briefly summarized below. A detailed description of both phases is found in the remainder of this section.

First, in the **issuance** phase, the customer acquires some tokens in exchange for payment. This phase consists of the following steps:

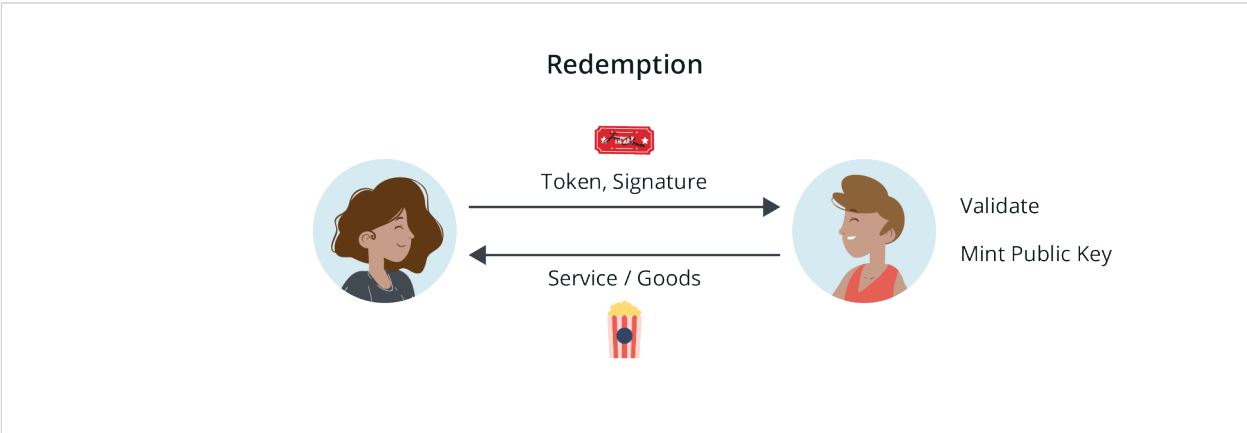
1. The user generates some **random** tokens.
2. The user **blinds** the tokens and sends the blinded tokens to the issuer for signing, along with a proof of payment.
3. The issuer returns **signatures** of the blinded tokens (equivalent to blinded signatures of the tokens) to the user. The blinding prevents the issuer from seeing the original tokens.

4. The user **unblinds** the signatures to reveal signatures of the original tokens. These are now ready to be exchanged for a service. No one apart from the user has ever seen these tokens, so they cannot be linked to the user's identity.



In the second step, the **redemption** phase, the user "spends" the tokens; that is, receives some service in exchange for the tokens. The following steps make up the redemption phase:

1. The user sends **the original token and the unblinded signature** to the service provider.
2. The service provider checks the validity of the signature.
3. The service provider also checks the **token number** to make sure it has not been "spent" yet. The token numbers are random, so they do not provide any information about the customer.
4. If everything is in order, the service provider **provides the requested service** to the user and records token as spent so that it cannot be used again.



In the remainder of this section, we review the issuance and redemption of ZKAPs in detail, providing the cryptographic details for each step. ZKAPs are based on Brave's implementation of Privacy Pass; we describe the evolution of the protocol and the differences between various implementations in section 3.

2.2 Cryptographic basics

Notation used in this chapter

Symbol	Meaning of Symbol	Example	Explanation
:	explanation of a variable	T : random token	T is used to signify the random token
==	equivalence	$P : Q == X : Y$	the relationship between P and Q is the same as between X and Y
:=	assigning a value to a variable	$x := 2y$	x is calculated by multiplying y by 2
=	equals	$P^k = T^{rk}$	P^k and T^{rk} have the same value
$\xleftarrow{\$}$	choose uniformly at random	$k \xleftarrow{\$} \mathbb{Z}_q$	k is chosen uniformly at random from $\mathbb{Z}_q$

The Cryptography of Zero-Knowledge Proofs

A zero-knowledge proof allows one entity (the prover) to prove the **knowledge** of a statement to another entity (the verifier) without revealing any information on the statement. In recent years, it has also become possible to do so in a **non-interactive manner**. In this use case, zero-knowledge proof cryptography allows the issuer of ZKAPs to prove that it signs blinded tokens correctly **without** revealing its secret key.

The ZKAPs implementation is based on Verifiable Oblivious Pseudorandom Functions (VOPRFs) using a Batched Discrete Log Equivalence proof (DLEQ). A DLEQ is a non-interactive zero-knowledge proof of knowlegde requiring that two pairs of values have the same discrete log relation:

$$\text{DLEQ}(P : Q == X : Y)$$

In other words, this proof confirms that for two pairs of values (P, Q) and (X, Y), if $Q = P^{k_1}$ and $Y = X^{k_2}$ are both true, then $k_1 = k_2$.

It is possible to batch such DLEQ proofs together to get a Batched DLEQ proof we call N-DLEQ. This increases efficiency, since only one DLEQ proof has to be generated for a whole batch of tokens, instead of a separate proof for every equivalence relation.

For the instantiation for ZKAPs, the Ristretto group is used.

Throughout this description of the protocol, we abstract the used group and use the following notation:

$\mathbb{G}$: group in multiplicative notation of prime order q and generator g
q : order of group $\mathbb{G}$
g : generator of group $\mathbb{G}$

Additionally, three hash functions are used. The three used hash functions are defined as follows:

$H_1 : \mathbb{Z}_q \to \mathbb{G}^*$, used in the issuance phase to create blinded tokens
$H_2 : \mathbb{Z}_q \times \mathbb{G} \to \{0,1\}^\kappa$, used in the redemption phase to create verification keys
$H_3 : \mathbb{G}^6 \to \mathbb{Z}_q$, used for the calculation of the batch DLEQ proof

Let us see how ZKAPs work in detail, going back to the carnival scenario. The cryptographic details for each step are shown in the purple boxes.

For a more detailed description of the protocol and the security guarantees, see the references in section 3.

First, the setting: a carnival is in town and publicly advertising itself. The poster bears the signature of Joe Clown, famous for organizing the best carnivals.
Everyone in town can see the advertisement. The audience can compare what they see over time and talk to each other about what they saw. They can tell they are in for a great time, because Joe Clown has a strong reputation, and his signature is a guarantee for quality.

k : secret key
Y : public key
g : generator of group
q : order of group

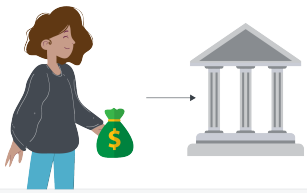
$k \xleftarrow{\$} \mathbb{Z}_q$
$Y := g^k$

The server generates a keypair consisting of a **secret key** k and the **public key** Y.

The secret key k is a random scalar that is only known to the server.

The generator g and order q of the group, as well as the public key Y, are known to all parties. The public key Y is the commitment by the issuer to a particular signing key. Customers can check that the public key remains unchanged over time, and does not vary between transactions. This guarantees that all ZKAPs are issued in the same manner, and that the issuance process will not introduce any identifiable information into the resulting ZKAPs.

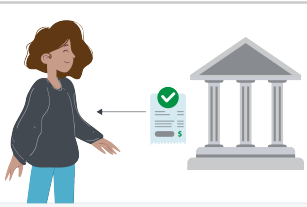
2.3 Issuance phase



Seeing the advertisement, Aditi decides to get some tickets.

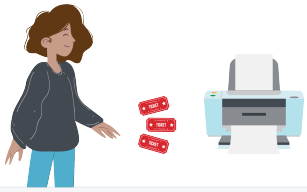
The customer purchases some ZKAPs using their preferred financial instrument. This could be any payment system – credit card, bank transfer, check – or it could be a different system entirely, for example, something like Zcash.

Not all of these systems provide anonymity, but this does not necessarily break the properties of ZKAPs, as we will see later.



Having paid, Aditi receives a proof of payment.

This can take different forms with different payment systems. Often, it is a printed or electronic invoice.



Next, Aditi prints out a group of her own random carnival tickets.
They don't contain identifying information related to Aditi or to other tickets in the group. She doesn't show these to anyone.

t : random bitstring/nonce
H_1 : hash function
T : random token
r : blinding factor
λ : security parameter
q : order of group

$t \xleftarrow{\$} \{0,1\}^\lambda$
$r \xleftarrow{\$} \mathbb{Z}_q$
$T := H_1(t)$

The user's client (e.g. an application or an extension) creates pairs of random nonces t and random blinding factors r. Both values t and r are chosen uniformly at random, t from bitstrings of length of the security parameter λ and r from $\mathbb{Z}_q$. Nonces in cryptography are arbitrary numbers, used only once. The random value t is called a `TokenPreimage` in this implementation.

Tokens T are derived from the random nonces t using the hash function H.

Please note that the tokens are generated in a batch. This means that not only one token is sampled at random and not only one blinding factor is generated, rather, these values are sampled in a batch. A batch consists of N tokens, so N random bitstrings/nonces and N blinding factors are sampled.

More concretely, they are calculated for all random nonces and random tokens:

$$t_1, \ldots, t_N \xleftarrow{\$} \{0,1\}^\lambda$$
$$r_1, \ldots, r_N \xleftarrow{\$} \mathbb{Z}_q$$
$$\text{for } i \in \{1, \ldots, N\} : T_i \coloneqq H_1(t_i)$$
$$\ldots$$

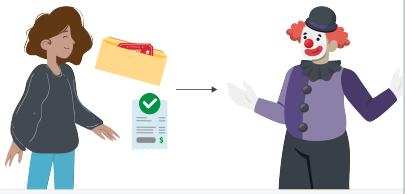
For simplicity, we follow the path of one token T in the batch, but keep in mind that all tokens are processed together as a batch for the DLEQ proof.

Then Aditi "blinds" the tickets she has printed:
She seals them inside a blank envelope beneath a piece of carbon paper.

T : random token
r : blinding factor
P : blinded token

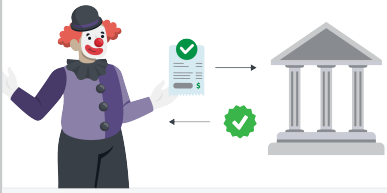
$$P \coloneqq T^r$$

Tokens are blinded to ensure that the issuer does not know which tokens they are signing. The blinded token is constructed using the random token T and the random blinding scalar r. The blinded tokens are denoted by P.



Now Aditi sends the sealed, blank envelopes and the proof of payment to the carnival's ticket master.

The user's client sends the blinded tokens and the proof of payment to the issuer's server.



The ticket master confers with the bank to validate the proof of payment.

The issuer of ZKAPs checks the validity of the proof of payment with the payment processor. The payment processor only has to refer to its transaction register to find the transaction identifier from the proof of payment.

The payment processor then tells the ticket master if the payment is real or not, but does not provide any identifying information about the customer.

Since the payment really was made, Joe Clown, the ticket master, proceeds to sign the blank envelopes.
The ticket master must sign the envelopes in a way that convinces Aditi that no tracking information has been added. We'll come back to this when Aditi is checking the signature.

Q : blinded signed token
P : blinded token
k : secret key
T : random token
r : blinding factor
N-DLEQ : DLEQ-Proof for a batch of $N + 1$ values

$$Q := P^k = T^{rk}$$
$$p := N\text{-DLEQ}(g, Y, \{P_i\}_i, \{Q_i\}_i)$$

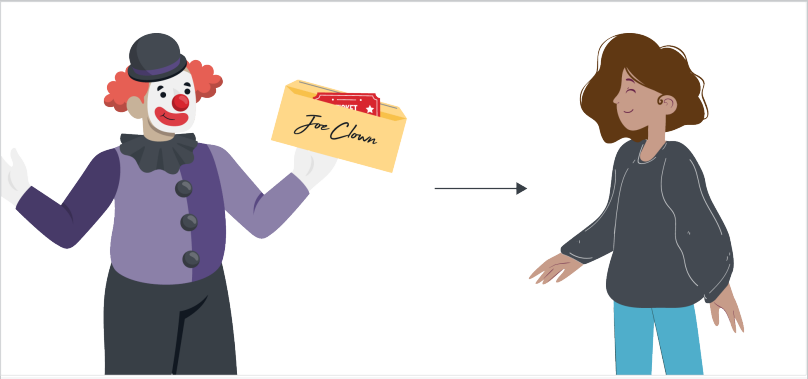
The server signs each blinded token P with its secret key k. The secret key, also called `SigningKey`, is never revealed to the client. The result is a `SignedToken` Q.

Please note that all earlier steps were conducted for multiple tokens T, resulting in multiple blinded tokens P and multiple blinded signed tokens Q. The server now also creates a proof (`BatchDLEQProof`) p showing that **all** blinded tokens were signed by the same secret key:

$$\text{DLEQ}(P_i : Q_i == P_j : Q_j)$$

That is, for all pairs of blinded tokens P_i and signed tokens Q_i for $i \in \{1, \ldots, N\}$, and for all pairs of blinded tokens P_j and signed tokens Q_j for $j \in \{1, \ldots, N\}$ if $Q_i = P_i^{k_i}$ and $Q_j = P_j^{k_j}$ are both true, then $k_i = k_j$ must also be true.

In this batching step, the third hash function H_3 is used.



Now the ticket master returns the signed, sealed envelopes to Aditi.

The user receives the signed tokens Q, along with the N-DLEQ proof p.

Next, Aditi inspects the signature. She makes sure it contains no extra information that might be used to track her.
She also compares the publicly visible signature on the poster to the signature she has received on the envelopes.
If Aditi thinks the signature on the envelopes deviates from the signature on the public poster, she shouldn't trust the tickets.

The public key that the issuer of ZKAPs uses must be visible to customers at all times. This ensures that the key is not changed from transaction to transaction, but all verifications use the same public key.

But let's suppose the signatures are good.
In this case, Aditi opens the envelopes and takes out the original tickets.
The carbon paper has copied the carnival signatures from the envelopes to the ticket each contained.
Aditi now holds tickets which no one else has ever seen, but which are signed in a way that only the carnival's ticket master could sign them.

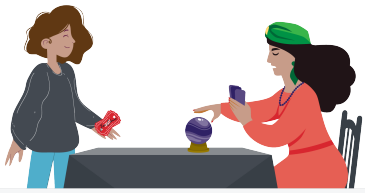
W : unblinded signed point
Q : blinded signed token
r : blinding factor
k : secret key
T : random token
t : random bitstring/nonce

$W := Q^{1/r} = T^k$
store (t, W)

The user unblinds the signatures using the random scalar r. This results in the unblinded signed point W, which, paired with the random preimage bitstring t, forms the unblinded token (t, W).

The tokens are now ready to be redeemed for some service or item.

2.4 Redemption phase



At last, Aditi makes her way to the fortune teller's booth.
She presents the signed tickets to the fortune teller.

t: random preimage bitstring
W: unblinded signed point
K: shared verification key
R: request
MAC_K: Message Authentication Code for shared key K
H_2 : hash function

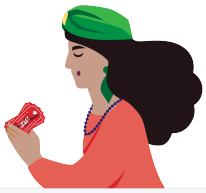
$K := H_2(t, W)$
$MAC_K(R)$: **Verification signature**
$(t, R, MAC_K(R))$: **Access Pass**

When the user wants to redeem the tokens, their client derives the shared verification key K from the unblinded token (t, W) using the second hash function H_2.

Using the verification key K and some request R (which could be a query to a web app or a piece of data to be stored), the user's client constructs the verification signature $MAC_K(R)$.

Finally, the ZKAP is composed of the random preimage bitstring t, the request R and the verification signature $MAC_K(R)$. The ZKAP is then sent to the server.

Note that the verification key K is **not** sent along with the ZKAP.



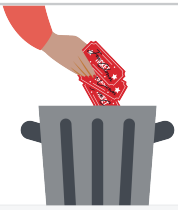
The fortune teller checks the tickets. Is the signature on the tickets really the ticket master's signature?

t: random preimage bitstring
R: request
k : secret key
MAC_K: Message Authentication Code for shared key K
H_1 : hash function
H_2 : hash function
T', W', K': recomputed random token, unblinded signed point, shared verification key

$T' := H_1(t)$
$W' := (T')^k$
$K' := H_2(t, W')$
$MAC_{K'}(R)$: **Verification signature**
Check $MAC_K(R) \stackrel{?}{=} MAC_{K'}(R)$

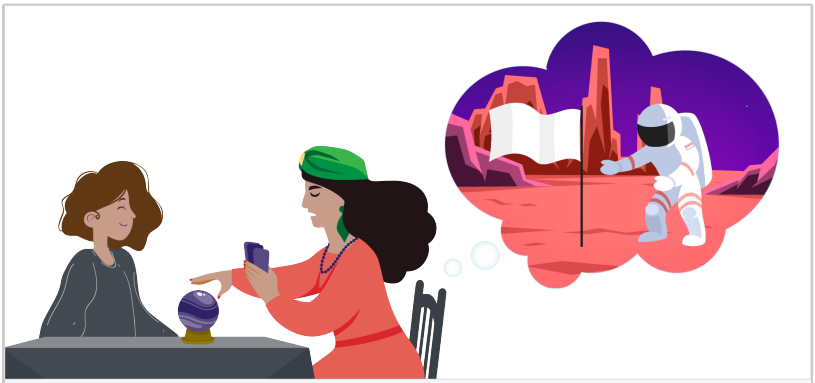
Upon receiving the ZKAP, the server derives the token T' from the token preimage t (contained in the ZKAP) and the hash function H_1. It then calculates the unblinded token W' using the token T' and the secret key k. Finally, it derives the signing key K' with the help of the second hash function H_2 and the unblinded token (t, W') it constructed.

Using the signing key K' derived above, the server can now construct a verification signature $MAC_{K'}(R)$, and compares it with the verification signature $MAC_K(R)$ it received as part of the ZKAP.



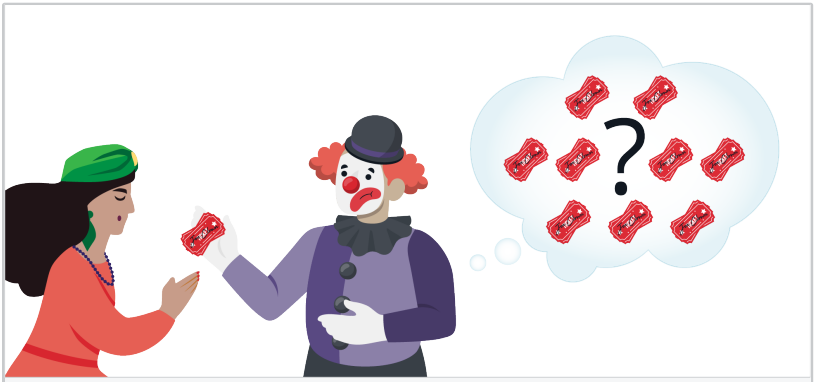
Finding the signatures to be valid, the fortune teller tosses the tickets in the trash so that Aditi can't use them again.

If the two verification signatures match, the server "knows" the ZKAP is valid. Now it checks whether the token has already been "spent" by checking the token preimage t against its list of past transactions.
If t has not yet been spent, the redemption succeeds and the server adds the token preimage t to the list of those already spent, so it cannot be redeemed twice.



And finally the fortune teller is happy to give Aditi her fortune. Luckily it is good news! With hard work and perseverance, Aditi's dream of being the first astronaut on Mars will come true.

The user is granted the request they paid for: access to some content, or storage of their data for a certain amount of time.



Here's a brief afterward to this story.
Aditi received quite a distinctive fortune and the fortune teller can't keep it to herself. She decides to tell the ticket master about it. The ticket master is really curious, so he'll try to find out who the first astronaut on Mars will be.

But when the fortune teller gives the ticket master the tickets associated with the fortune, there's nothing the ticket master can learn from them. The ticket master only remembers signing blank envelope after blank envelope, and has no idea which envelope contained these tickets.

And, since the ticket master always signs envelopes the same way, the signature on the ticket reveals no further clues.

The random blinding scalar r is not revealed to the issuer's server, so it cannot compute the unblinded token T from the blinded token P or the verification key K.
Even though the server receives the preimage bitstring t, it cannot connect this to a particular user or another preimage bitstring t', because t is a unique but **random** number.
This means that **the issuer of ZKAPs cannot deduce the user's identity** from the properties of the token, even if they have the intention to.

3

The evolution of ZKAPs

ZKAPs are based on Privacy Pass, an anonymous user-authentication mechanism that allows users to gain multiple tokens by solving a single CAPTCHA.

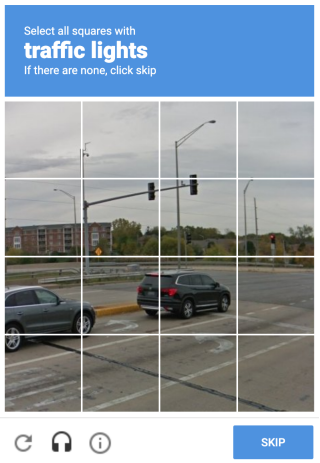
Brave developed a Rust implementation of the Privacy Pass protocol, which forms the basis of our implementation of ZKAPs.

ZKAPs use a slightly modified version of the Brave implementation to provide anonymous proof of payment for PrivateStorage, a privacy-preserving file storage system.

3.1 Privacy Pass

Privacy Pass (Davidson et al., 2018) is an anonymous whitelisting solution with a browser-based implementation that interacts with Cloudflare's server to store and "spend" tokens. It was developed to remedy the problem of certain groups of users having to solve too many CAPTCHAs when interacting with Cloudflare.

3.1.1 The problem with CAPTCHAs



Cloudflare, like many other Content Delivery Networks, filters IP addresses to prevent malicious attacks from the web (e.g. comment spam or SQL attacks). They do this by **IP reputation assessment**: trying to distinguish trustworthy IP addresses from malicious ones, which are often used by bots. The most common way to distinguish human users from bots is to make "suspicious" users solve a CAPTCHA .

Users of the Tor browser or VPN services were often mistaken for bots by Cloudflare. The reason for this is that Tor and VPN services do not show the provider the user's actual IP address, but they use one of their own IP addresses instead. Since this is a relatively small pool of IP addresses shared by many users, Cloudflare's security system often flagged these as suspicious.

As a result, Tor and VPN users were asked solve a CAPTCHA 17 times more often than other users (Davidson et al. 2018, p. 165), which impacted their user experience negatively.

3.1.2 One challenge, multiple tokens

The basic solution concept of Privacy Pass is that one action results in a batch of tokens: users only need to prove their humanness once, and get several tokens to use for several actions. In addition, Privacy Pass tokens do not compromise anonymity, as discussed in section 2.

Privacy Pass builds on Ecash by Chaum (1982, 1983), which provides a protocol for unlinkable token issuance and redemption using blinded tokens and blind signatures.

Privacy Pass uses an adaptation of the Oblivious Pseudorandom Function Protocol (OPRF) (Jarecki et al., 2014, 2016). This protocol allows users to ask for PRF evaluations from the provider holding the PRF key on inputs that are hidden from the provider. The OPRF protocol outputs a 1-RTT protocol for both signing and redemption.

Privacy Pass modifies Jarecki et al.'s non-interactive zero-knowledge proof of Discrete Log Equivalence (DLEQ), and uses a batch DLEQ proof for increased efficiency. This enables users to sign all their tokens with the same private key, and, in turn, prevents the provider from using different key pairs for different users and compromising their anonymity.

3.1.3 The Brave implementation

Brave has developed a Rust implementation of Privacy Pass using the Ristretto group. For implementation, Brave used the Merlin project.

3.2 Adapting Privacy Pass for PrivateStorage

ZKAPs	Privacy Pass
• Used as proof of payment • Integrated with Tahoe-LAFS • Tokens (ZKAPs) redeemed for storage time • Developed as a standalone app	• Used as proof of humanness • Integrated with Cloudflare • Personal information cannot be leaked or stolen – the service provider does not have it • Developed as a browser extension

ZKAPs were created by adapting Privacy Pass for a new use case: to provide an anonymous proof of payment for using them with PrivateStorage, a privacy-preserving file storage solution based on the Tahoe-LAFS storage system.

An open source, distributed secure storage solution.

Hosted and managed version of Tahoe-LAFS.

3.2.1 Anonymous file storage

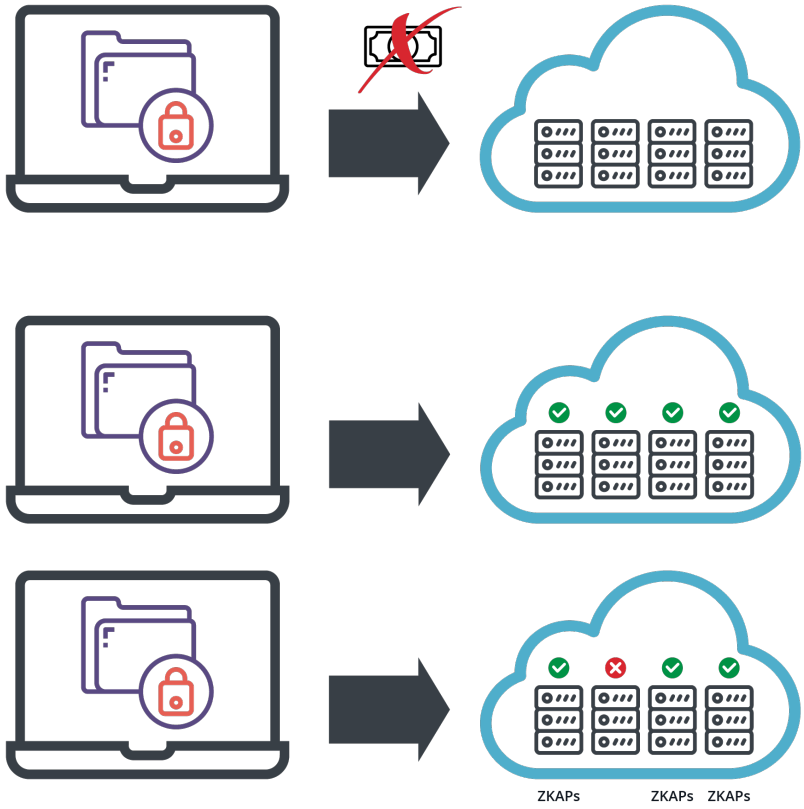
PrivateStorage offers a private, secure and end-to-end encrypted solution that aims to minimize the collection of any data related to its users. Developed by Least Authority, it is based on the Tahoe-LAFS secure storage system.

ZKAPs are generated and signed in batches, just like the tokens in Privacy Pass. However, instead of solving a CAPTCHA to prove one's humanness, ZKAPs require proof of payment to be validated.

PrivateStorage has a standalone app as the user's client. The ZKAPs are stored in the app, and redeemed when the user sends the data to the server to be stored.

Currently, 1 ZKAP is redeemed for 1 month's worth of storage of 1 megabyte of data in PrivateStorage.

3.2.2 Redeeming ZKAPs for anonymous file storage



Encrypted data becomes ciphertext, which is then divided into smaller parts called "shares". Shares of ciphertext get distributed on servers in a “grid” (collection of servers designated to store these data). The Tahoe-LAFS protocol does not require money for this to happen.

No user accounts means a different approach to pay-for-storage: we need a way to ensure that each share being received is paid for. The Tahoe-LAFS protocol includes leases on shares.

The PrivateStorage servers require ZKAPs to accept data for storing. The leases for shares are set based on the ZKAPs. Without ZKAPs, the storage servers will not allow the shares to be stored: if there is no proof of payment, the system will not provide the service (storing the data).

3.2.3 The advantages of PrivateStorage

ZKAPs can be used to authorize individual actions without identifying individual users, without the need for user accounts or access control lists. As demonstrated in section 2, tokens cannot be linked to token-holders or to each other.

This means that the service provider cannot create user "profiles" based on payment data or user accounts. So not only can providers not connect individual actions/data pieces to a user, they cannot even connect individual actions/data pieces to other actions/data pieces in the way they are able to do with account-based authentication – barring the caveats discussed in section 4.

Files and directories are encrypted on the client side (locally) using the encryption key held by the user. The provider does not have this key, so they don’t even know what the data they are storing is. This provides an extra dimension of privacy: not only is the personal information of the user hidden from the service provider, but the content of the files they are storing as well.

PrivateStorage also protects the integrity of the data stored on its servers: the provider cannot alter the data in any way. If the user’s client detects any modifications of the data coming from the server, it will not download the modified data.

The distribution of data as ciphertext shares provides improved availability. Any given piece of information is stored on more than one server. As a result, even if an entire server is unavailable (as a result of accidental damage or an attack), the data can still be recovered from other servers.

4

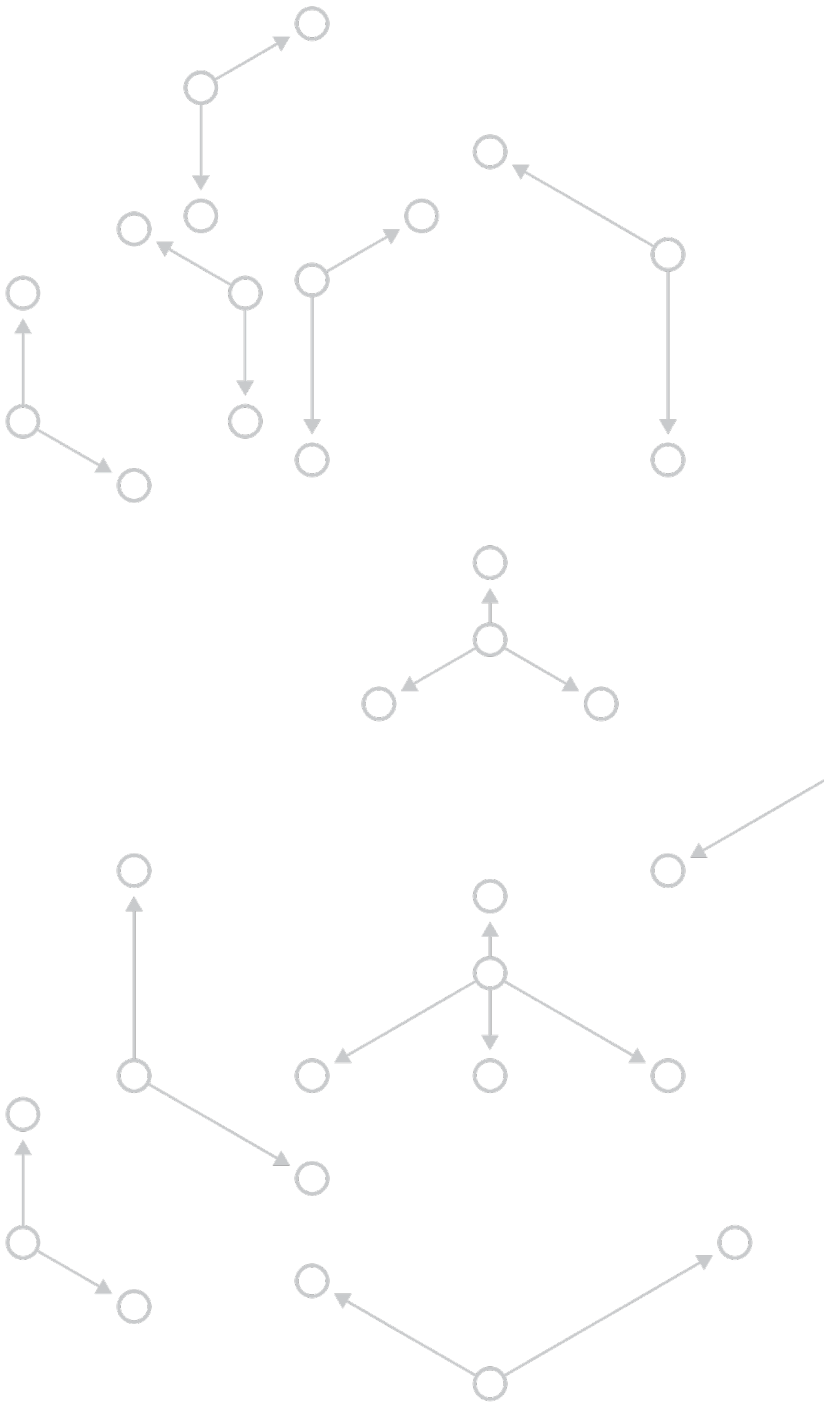
Known problems and limitations

Outside factors compromising privacy:

- Service providers can collect the meta-data of users (such as IP addresses or in-browser activity) to create profiles
- The less users a service has, the easier it is to connect in-app activity to a particular user

Limitations of the current implementation:

- There is only one denomination of ZKAPs
- The issuer and the service provider needs to be the same entity
- The issuer's signing key commitment needs to be strengthened



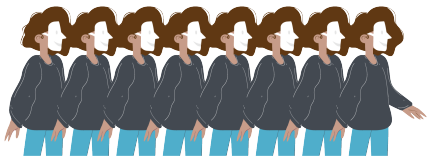
4.1 Outside factors influencing privacy

4.1.1 Metadata

Although ZKAPs do not reveal Aditi's identity, other metadata might: for example, facial recognition at the payment processor and throughout the carnival could easily identify her and track her actions.

On the internet, her IP address might be tracked from the payment processor to the point of service, and a profile could be created on the basis of her activity.

4.1.2 The necessity of a crowd



If Aditi is the only person at the carnival, no one needs to see her tickets to link her activities together. Aditi can disguise herself all she wants, but the carnival will still know it only issued one group of tickets in exchange for one payment. Other participants are necessary to provide deniability for any particular interaction.

In real life, running a service with only one user is bound to be unprofitable and unlikely to last; however, coupled with other information such as location or language settings, users can easily find themselves in a group of one, compromising their anonymity.

4.2 Limitations of the current implementation

4.2.1 Only one denomination

At the moment, all ZKAPs have the same value: they can pay for storing 1 megabyte of data for 1 month in PrivateStorage. As opposed to currencies like dollars or euros, where there are different denominations of bank notes ($1, $10, $100, etc.), there are no smaller "notes" for ZKAPs.

This means that tokens cannot be "divided" into smaller parts. If storing a megabyte for a **month** costs one token, there is no way to pay less than one token for storing a megabyte for a **day** or a **week**.

This encourages small-value tokens to avoid wasted value. However, the handling costs in the system scale at least linearly with the number of tokens issued and spent, so the value of the tokens cannot be too small – that would make the system uneconomical.

4.2.2 The issuer and service provider must be the same entity

When redeeming ZKAPs (see section 2.4), the verification of signatures requires the signing key. This means that any entity that can provide services in exchange for ZKAPs is also able to issue them. Consequently, the current protocol does not allow an intermediate level of privilege for entities that can "accept" ZKAPs but not issue them.

4.2.3 The issuer's signing key commitment

Anonymity in the system is provided by allowing users to hide in a crowd. All ZKAPs issued with the same signing key, which contributes to making the crowd larger. If the issuer uses multiple signing keys, then the crowd does not grow as large as it could. In the extreme, the issue could use a new signing key for each issuance and create many crowds, each containing one user, and defeat the anonymity properties of the system.

The issuer should make a public commitment to use a particular key, so that users know the issuer is not abusing the system in this way. By making the commitment in public, no individual user can be singled out to be placed in a smaller crowd. All users who see the public commitment will be members of the same cloud.

There are many possible ways to make such a public commitment. The PrivateStorage client will initially use a very simple mechanism in which one or more public keys are distributed with the client software. Potential users all receive the same client software, either directly from PrivateStorage's website or via sharing it with each other. Users can also check the cryptographic signature of the client, which is also available from the PrivateStorage website. By virtue of receiving the same software, users also receive the same keys.

In the future we hope to extend this to a system which allows convenient, anonymity-preserving key rotation over time.

5

Expanding the use of ZKAPs

The advantages of using ZKAPs:

- Reducing the risk and overhead related to handling personal data by companies
- Integrating services without sharing customer data
- Attracting privacy-conscious customers

Potential use cases:

- Digital invitation systems
- Public transport payments
- VPN and messaging services
- Content escrow using 2 types of ZKAPs

Although we created ZKAPs to better address the access-control issue in Tahoe-LAFS for the development of PrivateStorage, we see many possibilities for the use of ZKAPs to help protect user privacy in other services that need to accept online payments or perform authorization based on other criteria (e.g. volunteer groups, educational institutions). We discuss some of these potential use cases in this section.

5.1 The advantages of adopting ZKAPs

The use of ZKAPs can help facilitate an online exchange of value while disconnecting the payment and service data that is gathered on customers. This is very helpful in use cases where mixing these data points is not in the best interest of the company offering the service. While collecting personal data can be incredibly valuable to some services ("data is the new oil") it can just as often be a liability to others ("data is toxic waste").

5.1.1 Reducing the risks of handling customer data for service providers

Two of the biggest risks in connection with collecting data are privacy regulations and data breaches. Under GDPR (General Data Protection Regulation), the fines alone can amount to to €20 million, or 4% of a company's annual global turnover. Data breaches, on the other hand, cause serious damage to the reputation of companies. Since 2014, there have been 7 security incidents at major companies that resulted in the Chief Information Security Officers losing their jobs – Equifax, Facebook and Uber, to name a few.

Decoupling payment from service data greatly reduces the data breaches and compliance challenges.

5.1.2 Payments interoperability: allowing services to integrate without sharing customer data

Different payment processors may find it unattractive to share their customer data with other payment processors in order to interoperate, so they are not incentivized to do so.

ZKAPs can offer an approach to securely and privately facilitate interoperability by utilizing tokenization to prevent the need to share customer data with the other payment processors. This allows payment processors to offer more flexibility to their customers (interoperable payments) without risking the exposure of customer data, and provides an advantage over competitors.

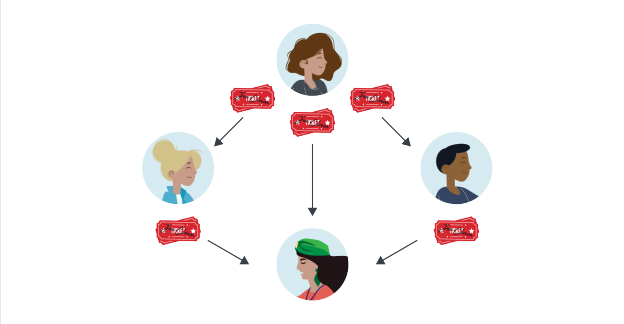
5.1.3 Increasing customer satisfaction

In addition to its advantages for service providers, disconnecting payment data from service data can offer value to customers. The company offering a service can still know who its customers are through payment data. However, customers may not want that company to know **how** they use the service—specifically, for the company to tie behavior that they observe (service data) to an individual name. This can be relevant for file storage services, but also for any other kind of use that may be privacy-sensitive, such as medical advice or even newspaper consumption.

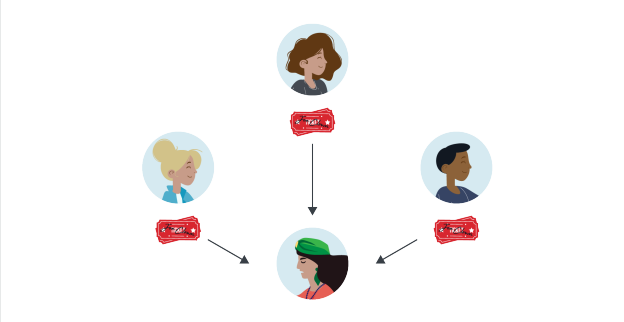
In the remainder of this section, we discuss some potential use cases for ZKAPs.

5.2 Potential use cases for ZKAPs

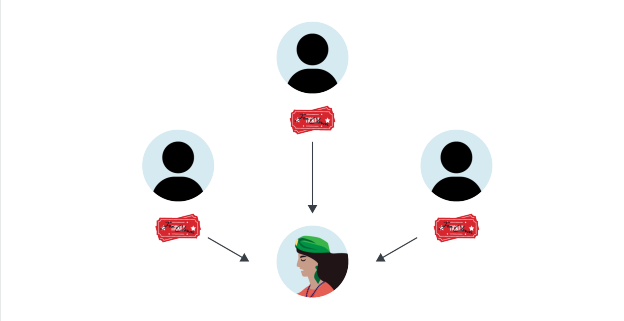
5.2.1 Digital gift cards and invitation systems



Services trying to build a user base often want to encourage users to invite their friends to use the service. This often happens via personalized links or gift cards (in the form of personalized codes), revealing users' social network graph to the service.



As a privacy-preserving solution, the service could issue a number of access passes like ZKAPs which users could distribute to their friends to redeem in exchange for joining the service. Since ZKAPs's token numbers are random, the service provider cannot connect new user to the one who gave them the pass. This "breaks the edges" in the social network graph from the perspective of the service provider.



If the service also uses ZKAPs for authorization instead of an identity-based scheme, it could also avoid collecting users' identities entirely. This removes identifying information from the nodes of the social network graph, which is now less of a graph and more of a bland point cloud.

5.2.2 Public transport

Another use case is payment for a physical service, such as using a public transit system. Many such systems have transitioned to digital payment schemes in recent years. Using identity-based authorization, however, provides the opportunity to track individual's movements through the transit network, and, typically, to link these movements to a customer's real identity.

A system utilizing ZKAPs would still allow for the digital processing of payments without the possibility for tracking the physical movement of users.

5.2.3 VPN and direct messaging services

Privacy-focused VPN (Virtual Private Network) and direct messaging apps have seen a rising demand: not only do users wish to stop their data from being commercialized, there is a rising pushback against privacy from repressive governments. While the data going through the service are usually well-protected, personal information can still be leaked via logs or billing. In some countries, merely using such privacy-focused services can have repercussions, and providers can be ordered to reveal such data about their users.

ZKAPs can give a company the ability to store service usage logs without having to worry about personal data appearing in those logs. Since ZKAPs do not reveal the identity of users, **service providers cannot hand over these data** even if ordered to do so by a government body.

At the same time, the fact that someone is using certain VPN or messaging services can potentially be revealed by the data stored by the **payment provider**. Employing an anonymous payment method for purchasing ZKAPs decreases the likelihood of this scenario.

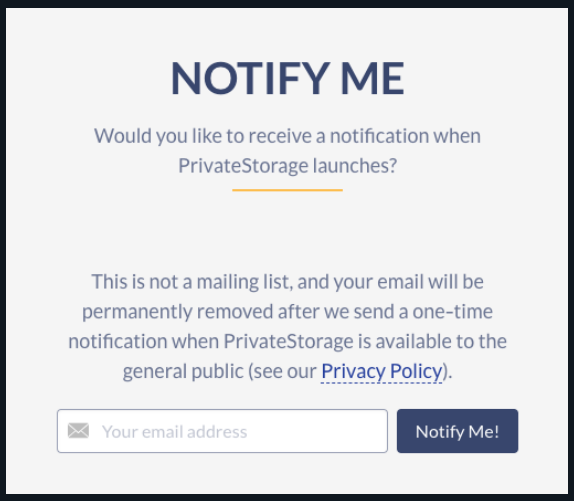
5.2.4 Content escrow

Monetizing digital assets typically requires that customers reveal personal information to the digital asset creator, the file hosting entity and/or a payment processor. A file escrow system operating between digital asset producers and purchasers, utilizing Tahoe-LAFS and ZKAPs could facilitate an online exchange of value while disconnecting the payment and service data.

This system could use 2 types of ZKAPs to act as access control to both uploading and downloading files. The uploading of files (by authors or publishers) would be based on a proof of copyright or proof of creation (type 1 ZKAPs), while a second type of ZKAPs would act as proof of payment to allow users to download files from the Tahoe-LAFS storage servers (type 2 ZKAPs).

In order to provide the necessary gatekeeping for the value exchange, we need to check on proof of creation for the digital asset supplier and proof of payment by the digital asset purchaser. Both the proof of creation and proof of payment are real world interactions that would need to be confirmed by trusted third parties. After the proof of creation is verified by a trusted party, type 1 ZKAPs would be issued, allowing access to upload and store data on the Tahoe-LAFS storage nodes. These are the ZKAPs currently implemented: they allow for upload, and they are redeemed upon the uploading of the files.

On the other side, the proof of payment would result in the issuance of a new type of ZKAPs (type 2), that allow for access to download the files, at which point they are redeemed.



ZKAPs in Action

- We will be launching PrivateStorage later this year! Sign up to be notified at https://private.storage/
- We are investigating offering ZKAPs as a standalone service.
  Email us if you want to talk about using ZKAPs at contactus@leastauthority.com

References

Chaum, D. (1982). Blind Signatures for Untraceable Payments. In D. Chaum, R. L. Rivest, and A. T. Sherman (Eds.), *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*, pp. 199–203. Plenum Press, New York.

Chaum, D. (1983). Blind signature system. In D. Chaum (Ed.), *Advances in Cryptology, Proceedings of CRYPTO '83, Santa Barbara, California, USA, August 21-24, 1983*, pp. 153. Plenum Press, New York.

Davidson, A., I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda (2018). Privacy Pass: Bypassing Internet Challenges Anonymously. *Proceedings on Privacy Enhancing Technologies 2018*, 164 – 180.

Jarecki, S., A. Kiayias, and H. Krawczyk (2014). Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model. In P. Sarkar and T. Iwata (Eds.), *Advances in Cryptology – ASIACRYPT 2014*, Berlin, Heidelberg, pp. 233–253. Springer Berlin Heidelberg.

Jarecki, S., A. Kiayias, H. Krawczyk, and J. Xu (2016). Highly-Efficient and Composable Password-Protected Secret Sharing (Or: How to Protect Your Bitcoin Wallet Online). *IACR Cryptology ePrint Archive 2016*, 144.

Links

- Brave's Rust implementation of Privacy Pass.
- Merlin: a transcript construction for zero-knowledge proofs.
- PrivateStorage: a privacy-enhanced cloud storage product based on Tahoe-LAFS.
- Ristretto: a technique for constructing prime order elliptic curve groups with non-malleable encodings.
- Tahoe-LAFS: a free and open decentralized cloud storage system.