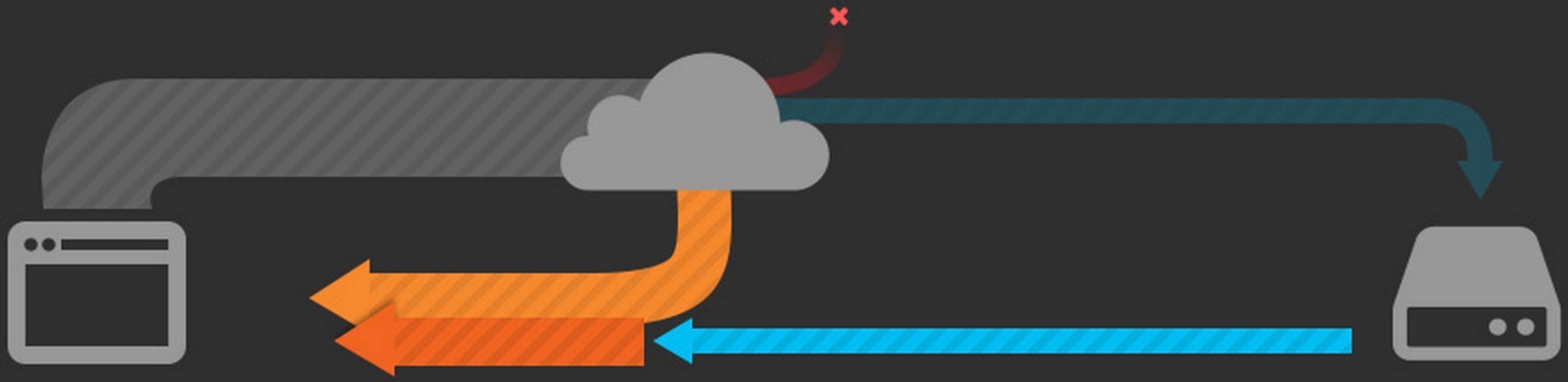
PRIVACY PASS

NICK SULLIVAN
CLOUDFLARE
@GRITTYGREASE

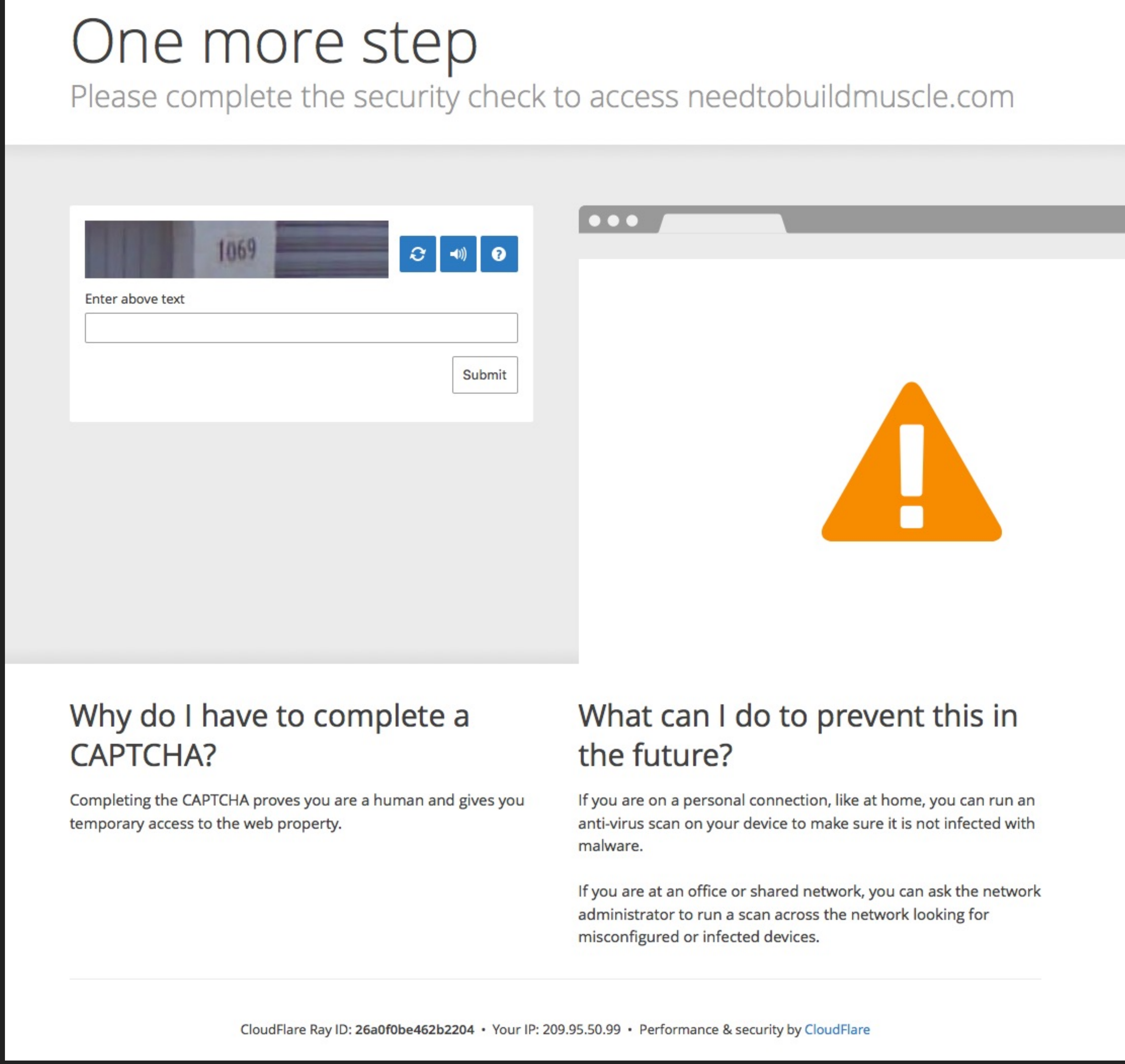
A LIGHTWEIGHT ZERO-KNOWLEDGE PROTOCOL

Cloudflare Reverse Proxy



THE PROBLEM OF SCALE

- Reducing malicious activity online
  - CAPTCHA challenge for risky requests
  - Issue a cookie once cleared
- Disproportionately affects privacy-conscious
  - VPN, Tor users share IPs with bad actors resulting in bad IP reputation
- Cookies are not portable
  - Web origin policy prevents tracking



Global

- Cisco
- Eurovision Song Contest
- Dish Network
- IBM
- Warner Bros.
- Pfizer
- Sennheiser
- Hertz
- Uber
- RTÉ
- Johnson & Johnson
- Novartis
- GE
- NYSE
- Riot Games
- zendesk

Financial Services

- Goldman Sachs
- Banc of California
- Montecito Bank & Trust
- Nasdaq
- Capital Bank
- Personal Capital
- Symphony
- Bci somos diferentes
- Autorité des marchés financiers
- BancoEstado
- coinbase
- TradeKing
- TSB
- acorns
- Bitfinex
- BDO

Public Sector

- GPO
- Trump Pence
- az.gov State of Arizona
- Department of State
- City of Baltimore
- National Cyber Security Centre
- Central Bank of Nigeria
- State of New Hampshire
- Federal Bureau of Investigation
- isle of man
- Library of Congress
- www.parliament.uk
- Australian Government
- City of Flint Michigan

Technology

- Cisco
- vmware
- box
- xerox
- AT&T
- zendesk
- Curse
- glassdoor
- Plex
- yelp
- Discord
- Lattice
- okcupid
- Parallels
- LogMeIn
- Dropbox

eCommerce

- zumiez
- Steve Madden
- Redbubble
- Bose
- Nixon
- Casper
- Lush
- Steelers
- Sur la table
- Warby Parker
- redbox.
- Snapple
- thriftbooks
- fitbit
- fiverr
- Lootcrate

11M+

websites, apps & APIs
in 150+ countries

CLOUDFLARE

WOULDN'T IT BE NICE TO HAVE AN ONLINE EQUIVALENT TO CASH?

- Desirable properties
  - Withdrawals and transactions are un-linkable
  - Can only be created by a central authority

CASH IS NOT ANONYMOUS IN A PANOPTICON SCENARIO


Serial Number: 00003304043030


Serial Number: 00003304043030


Official Bill


Official Bill


Serial Number: 00003304043030
Official Bill

Serial Number: 000033040430 30

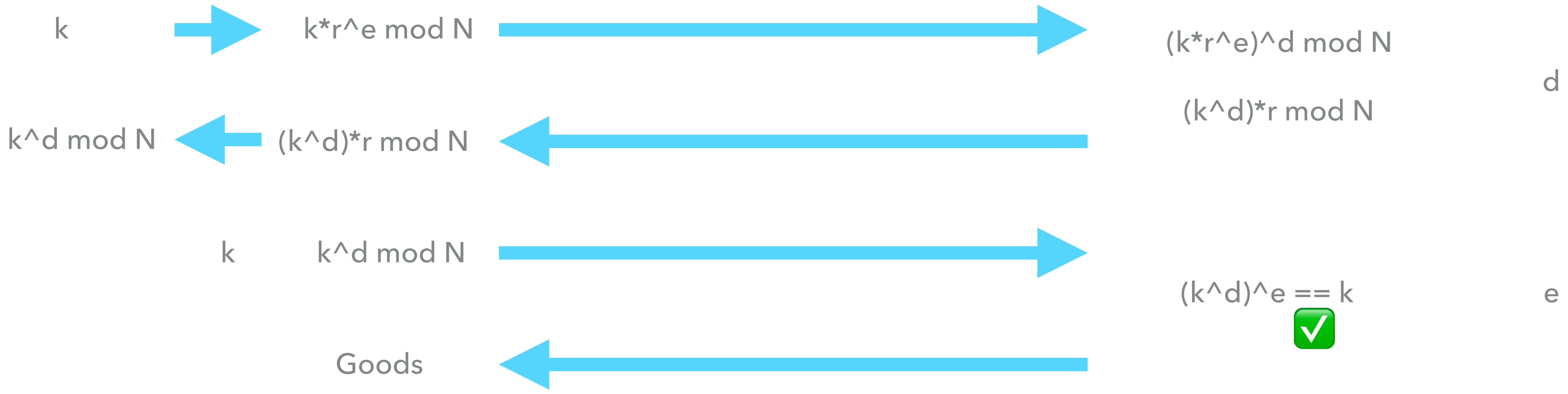
Official Bill

ECASH (CHAUM, 1983)

- Digital version of cash based on blind RSA signatures

ECASH (CHAUM, 1983)

- Digital version of cash based on blind RSA signatures



Filename: captcha-plugin-draft.txt
Title: Toward a better Cloudflare CAPTCHA
Authors: George Tankersley, Filippo Valsorda
Created: 19-Jan-2016
Status: Draft

Change history:
  2016-01-19: Initial draft
  2016-02-06: Revisions, filled in some TODOs
  2016-02-12: Fill in more blanks

Overview:

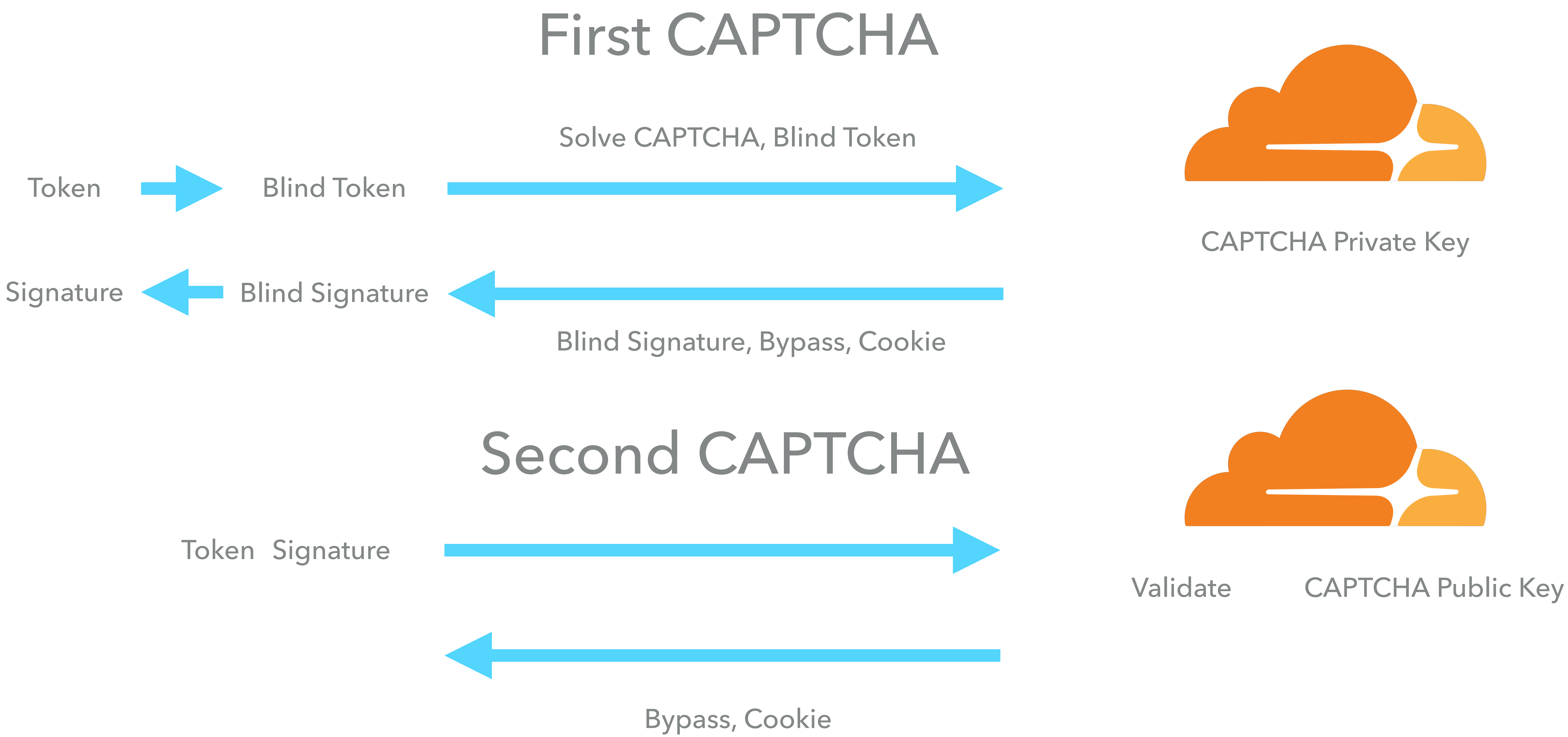
In many IP reputation systems, Tor exits quickly become associated with abuse. Because of this, Tor Browser users quickly become familiar with various CAPTCHA challenges. The worst of these is Cloudflare- since their CAPTCHA service requires JavaScript to serve anything but unreadable noise, they render an increasingly large portion of the internet inaccessible to Tor Browser users.

This document describes a solution to this problem. We propose a Tor Browser plugin that will store a finite set of unlinkable CAPTCHA bypass tokens and a server-side redemption system that will consume them, allowing users to forego solving a CAPTCHA so long as they supply a valid token.

0.1. Rationale

Since the Cloudflare system depends heavily on an IP reputation database to detect abuse, these challenge pages have become a familiar sight to users of Tor, I2P, and popular VPN services. While they are intended as a minor inconvenience, they are impenetrable for users with high privacy or security requirements. ReCAPTCHA de facto demands JavaScript execution- the challenges it produces without JS have degraded to such an extent that humans frequently cannot solve them. Worse, the challenge page sets a unique cookie to indicate that the user has been verified. Since Cloudflare controls the domains for

ECASH AS A CAPTCHA SOLUTION



IS THIS IT?

- It wasn't satisfying to use slow 1980s cryptography
- There have been recent constructions that to similar things to ecash
- OPRF: Oblivious Pseudo-Random Function
  - Analogous to blinding
- VRF: Verifiable Random Functions
  - Random function provably computed with a private key
- VOPRF: Verifiable OPRF with VRF-like confirmation phase (new)

INSPIRATIONS/PRIOR WORK

- Freedman et al. (2005) seems to be the first to construct an OPRF (with a constant-number of rounds)
- This was extended by Jarecki et al (2009) for performing set intersection functionality
- The work of Jarecki, Kiayias and Krawczyk (2014) presents a VOPRF that is very similar to our construction (without the batched DLEQ proofs). They use it to construct round-optimal password-protected secret sharing (PPSS) and T-PAKE.

INSPIRATIONS/PRIOR WORK

- Shirvanian et al. construct SPHINX: A Password Store that Perfectly Hides Itself from OPRFs. The OPRF used is essentially the same as the one we use, except it has no verifiability and the input data is structured differently.
- Elliptic-curve based random functions: Golberg et al. present a verifiable random function (without blinding and without batched DLEQ proofs).
- Burns et al. (2017) give an explicit EC instantiation of the OPRF that we use, except without the DLEQ proof (i.e. no verifiability).
- Ryan Henry's PhD thesis (2014) describes the batched DLEQ proof we use.

FUNDAMENTAL COMPONENTS / TERMINOLOGY

- Prime order group
  - e.g. The group of points on an Elliptic Curve such as P-256
  - Group elements will be denoted by capital letters such as P or Q
- Scalar multiplication
  - Adding a point to itself n times, such as P+P+…+P is denoted nP
  - Scalars will be represented by lower-case letters

FUNDAMENTAL COMPONENTS / TERMINOLOGY

- Hash to group element
  - Function that takes a scalar and outputs a random group element
- Discrete log equivalence proof
  - A short zero-knowledge proof that two pairs of points have the same discrete logarithm. For example, (P, sP) and (Q, sQ) have the same discrete logarithm. The same scalar s is used for P and Q to get sP and sQ.
  - Denoted DLEQ(P:R == Q:S)

Scenario 1

The client takes a point on an elliptic curve `T` and sends it to the server. The server applies a secret transformation (multiplication by a secret number `s`) and sends it back. Call this step “Issue”, as the server issues a signed point to the client.

Issue

```
T ->
  <-  sT
```

Later, the client sends `T` and `sT` to the server to prove it has previously issued `sT`.

Redeem

```
T, sT ->
```

Since only the server knows `s`, it can confirm that it had issued `sT`. We call this step “Redeem”.

Problem: Linkability

In this situation, the server knows `T` because it has seen it already. This lets the server connect the two requests, something we’re trying to avoid. This is where we introduce the blinding factor.

Scenario 2

Rather than sending `T`, the client generates its own secret number `b`. The client multiplies the point `T` by `b` before sending it to the server. The server does the same thing as in scenario 1 (multiplies the point it receives by `s`).

Issue

```
bT ->
   <- s(bT)
```

The client knows `b` and `s(bT)` is equal to `b(sT)` because multiplication is commutative. The client can compute `sT` from `b(sT)` by dividing by `b`. To redeem, the client sends `T`, `sT`.

Redeem

```
T, sT ->
```

Since only the server knows `s`, it can confirm that `sT` is `T` multiplied by `s` and will verify the redemption.

Problem: Malleability

It's possible to create an arbitrary number of pairs of points that will be verified. The client can create these points by multiplying both `T` and `sT` by an arbitrary number `a`. If the client attempts to redeem `aT` and `a(sT)`, the server will accept it. This effectively gives the client unlimited redemptions.

Scenario 3

Instead of picking an arbitrary point `T`, the client can pick a number `t`. The point `T` can be derived by hashing `t` to a point on the curve using a one-way hash. The hash guarantees that it's hard to find another number that hashes to `aT` for an arbitrary a.

Issue

```
T = Hash(t)
bT ->
    <- sbT
```

Redeem

```
t, sT ->
```

Since only the server knows s, it can compute `T = Hash(t)` and confirm that `sT` is `T` multiplied by `s` and will verify the redemption.

Problem: Redemption hijacking

If the values `t` and `sT` are sent across an unsecured network, an adversary could take them and use them for their own redemption.

Sending `sT` is what lets attackers hijack a redemption. Since the server can calculate `sT` from `t` on its own, the client doesn't actually need to send it. All the client needs to do is prove that it knows `sT`. A trick for doing this is to use `t` and `sT` to derive a HMAC key and use it to sign a message that relates to the redemption. Without seeing `sT`, the attacker will not be able to take this redemption and use it for a different message because it won't be able to compute the HMAC key.

Scenario 4

Instead of sending `t` and `sT` the client can send `t` and `HMAC(sT, M)` for a message `M`. When the server receives this, it calculates `T = Hash(t)`, then uses its secret value to compute `sT`. With `t` and `sT` it can generate the HMAC key and check the signature. If the signature matches, that means the client knew `sT`.

Issue

```
T = Hash(t)
bT ->
    <- sbT
```

Redeem

```
t, M, HMAC(sT, M) ->
```

Since only the server knows s, it can compute `T = Hash(t)` and compute `sT` as `T` multiplied by `s` and verify the HMAC to validate that the client knew `sT`.

Problem: Tagging

The server can use a different s for each client, say `s_1` for client 1 and `s_2` for client 2. Then the server can identify the client by comparing `s_1*Hash(t)` and `s_2*Hash(t)` against the `sT` submitted by the client and seeing which one matches.

Scenario 5

The server picks a generator point `G` and publishes `sG` somewhere where every client knows it.

Issue

```
T = Hash(t)
bT ->
    <- sbT, DLEQ(bT:sbT == G:sG)
```

The client can then check to see that the server used the same `s`, since everyone knows `sG`.

Redeem

```
t, M, HMAC(sT, M) ->
```

Just like in Scenario 4, since only the server knows s, it can compute `T = Hash(t)` and compute `sT` as `T` multiplied by `s` and verify the HMAC to validate that the client knew `sT`.

Problem: only one redemption per issuance

This system seems to have all the properties we want, but it would be nice to be able to get multiple points.

Scenario 6

The client picks multiple values `t1, t2, … , tn` and multiple blinding factors `b1, b2, … , bn`.

Issue

```
T1 = Hash(t1)
T2 = Hash(t2)
T3 = Hash(t3)
b1T1 ->
b2T2 ->
b3T3 ->
      <- sb1T1, DLEQ(b1T1:sb1T1 == G:sG)
      <- sb2T2, DLEQ(b2T2:sb2T2 == G:sG)
      <- sb3T3, DLEQ(b3T3:sb3T3 == G:sG)
```

Each DLEQ can be verified independently like in Scenario 4, the client is safe from tagging.

Redeem

```
t1, M, HMAC(sT1, M) ->
```

This lets the client do multiple redemptions.

Problem: Bandwidth

DLEQ proofs are not particularly compact. Luckily, they can be optimized with something called an efficient batch DLEQ proof. It’s essentially a single proof that covers all the returned values. This can be done by computing a proof over a random linear combination of the points:

Because the same `s` is used for every `T`, you can use the commutative property of multiplication again to help you.

```
sb1T1+sb2T2+sb3T3 = s(b1T1+b2T2+b3T3)
```

So the server can compute a single DLEQ that proves that the same s was used for each T:

`DLEQ(b1T1+b2T2+b3T3:s(b1T1+b2T2+b3T3) == G: sG)` This is the same size as a single DLEQ proof.

Scenario 7

This scenario is similar to the last one except that the server sends a batch DLEQ proof rather than one for each point.

Issue

```
T1 = Hash(t1)
T2 = Hash(t2)
T3 = Hash(t3)
b1T1 ->
b2T2 ->
b3T3 ->
    c1,c2,c3 = H(G,sG,b1T1,b2T2,b3T3,s(b1T1),s(b2T2),s(b3T3))
    <- sb1T1
    <- sb2T2
    <- sb3T3
    <- DLEQ(c1b1T1+c2b2T2+c3b3T3:s(c1b1T1+c2b2T2+c3b3T3) == G:sG)
```

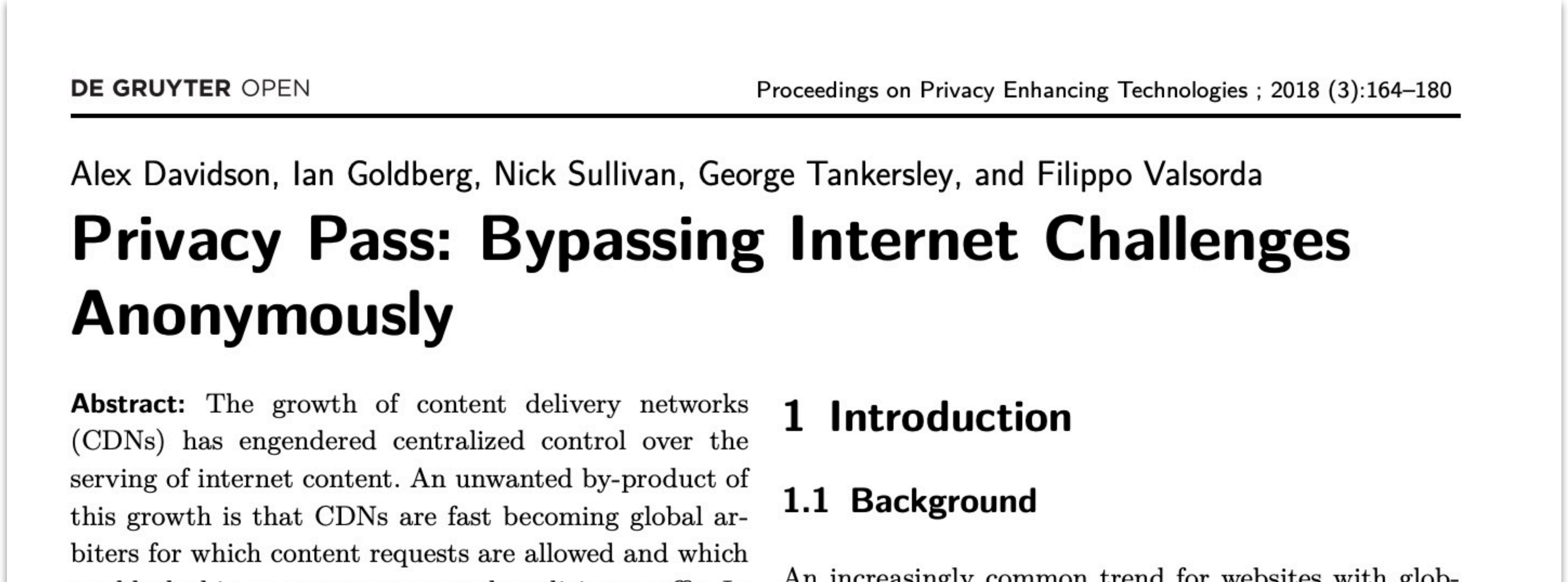
This DLEQ proof can be validated by recomputing `z = c1,c2,c3` and then `c1b1T1+c2b2T2+c3b3T3` and `sc1b1T1+sc2b2T2+sc3b3T3`.

Redeem

```
t1, M, HMAC(sT1, M) ->
```

IS THIS IT?

- For the protocol, yes!
- But we also released Privacy Pass as a Firefox and Chrome extension
- ~50,000 daily active users
- Trillions of requests per week



DE GRUYTER OPEN Proceedings on Privacy Enhancing Technologies ; 2018 (3):164–180

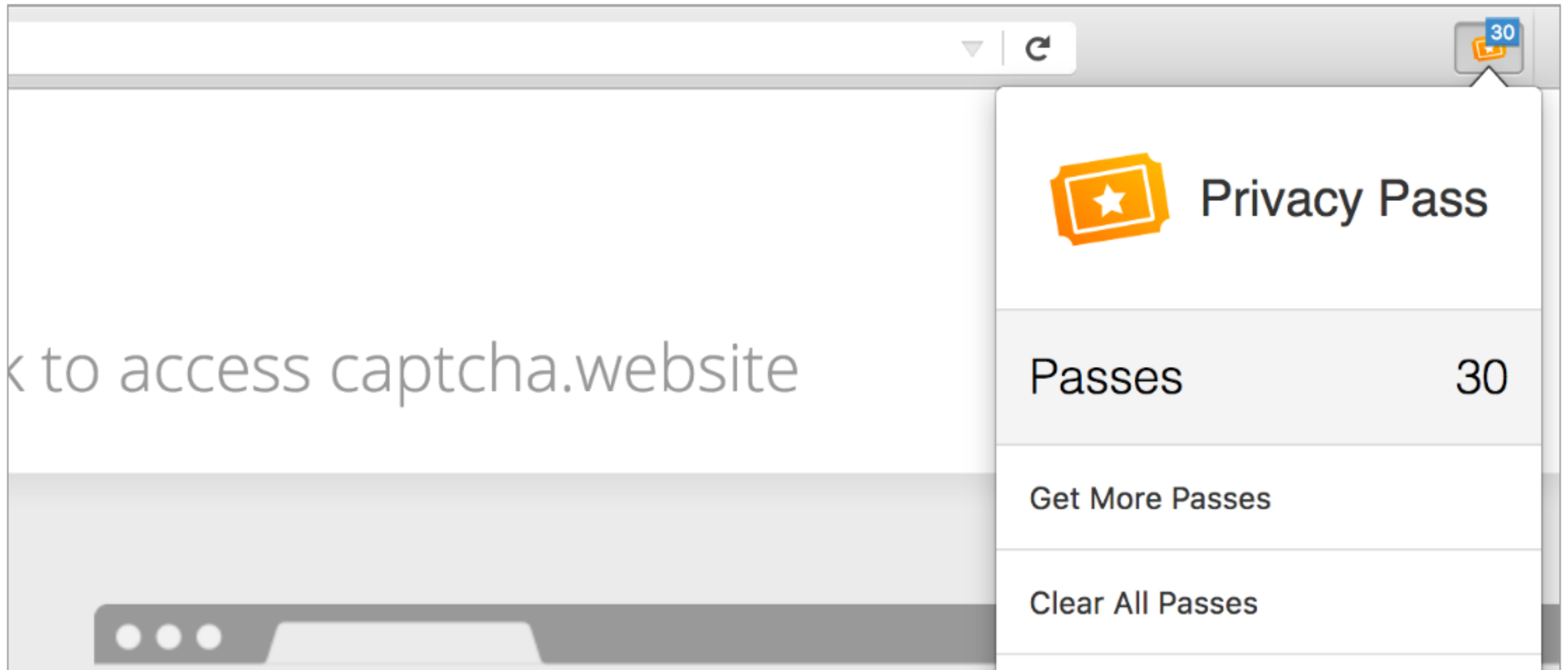
Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda

Privacy Pass: Bypassing Internet Challenges Anonymously

Abstract: The growth of content delivery networks (CDNs) has engendered centralized control over the serving of internet content. An unwanted by-product of this growth is that CDNs are fast becoming global arbiters for which content requests are allowed and which

1 Introduction

1.1 Background



LOOKING FORWARD

- Currently integrating Privacy Pass with more CAPTCHA providers
  - VOPRF is not publicly verifiable, more like a voucher than cash
- (V)OPRFs proposal submitted to CFRG
- Additional applications of the idea
  - Anonymous session resumption for TLS
  - Anonymous referral code mechanism
  - Single bit zero-knowledge proofs (am I over 18? etc.)

PRIVACY PASS

NICK SULLIVAN
CLOUDFLARE
@GRITTYGREASE

NIZK Discrete Log Equality

$$\log_G(Y) =? \log_M(Z)$$
$$(Y = kG, Z = kM)$$

$\mathsf{DLEQGenerate}(G, Y, M, Z)$	$\mathsf{DLEQVerify}(G, Y, M, Z, (c, s))$
$r \leftarrow \mathbb{Z}_p$	$A' = sG + cY$
$A = rG$	$B' = sM + cZ$
$B = rM$	$c' = H(G, Y, Z, A', B')$
$c = H(G, Y, Z, A, B)$	Output $c == c'$
$s = (r - ck)(\mathrm{mod} p)$	
Output (c, s)	

Jarecki, Kiayias, Krawczyk. Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only model.