# Fast Privacy-Preserving Punch Cards


ANNETTE'S CAFÉ

# Blend Eatery

Inbox ×

**Jeeryn Dang** [redacted]
to saba, Marc

4:07 PM (3 minutes ago)

Hello Saba,
Thank you for your email and checking in.  We appreciate it tremendously.

We are currently still closed and don't have a tentative date to open yet. At the moment, we are waiting to hear back from Stanford as to when we will be able to get the green light.  From what we understand, research Is slowly ramping back up in phases. We were told that we may be able to open back up once research does, but there are still a lot of details that need to be ironed out before we can do so.

We are currently servicing hospital on Wednesdays and Thursdays via a pop-up location at their old cafe.  This has allowed us to at least employ some of our staff.

Please don't hesitate to reach out if you have any additional questions. We look forward to being able to open back up and once again serving the University community.  We miss you guys!

Cheers,
Jeeryn and Marc

BUY 9 BOWLS, GET THE 10TH ONE FREE!


*(CARD WILL BE PUNCHED FOR BUILD-A-BOWLS ONLY. COMPLETED PUNCH CARD CAN BE REDEEMED FOR A FREE BOWL WITH A SINGLE PROTEIN CHOICE AND FRIED EGG. ANY EXTRAS WILL BE AN ADDITIONAL COST)

BLENDEATERY.COM | @BLENDEATERY 382 NORTH SOUTH AXIS STANFORD, CA 94305


BUY 9 BOWLS, GET THE 10TH ONE FREE!
YUM!
*(CARD WILL BE PUNCHED FOR BUILD-A-BOWLS ONLY. COMPLETED PUNCH CARD CAN BE REDEEMED FOR A FREE BOWL
WITH A SINGLE PROTEIN CHOICE AND FRIED EGG. ANY EXTRAS WILL BE AN ADDITIONAL COST)
BLENDEATERY.COM | @BLENDEATERY 382 NORTH SOUTH AXIS STANFORD, CA 94305


## Why go digital?

Customer convenience
No lost cards
Better bookkeeping
Hard to Counterfeit
Contactless


<u>Why go digital?</u>

Customer convenience
No lost cards
Better bookkeeping
Hard to Counterfeit
Contactless


## Why go digital?

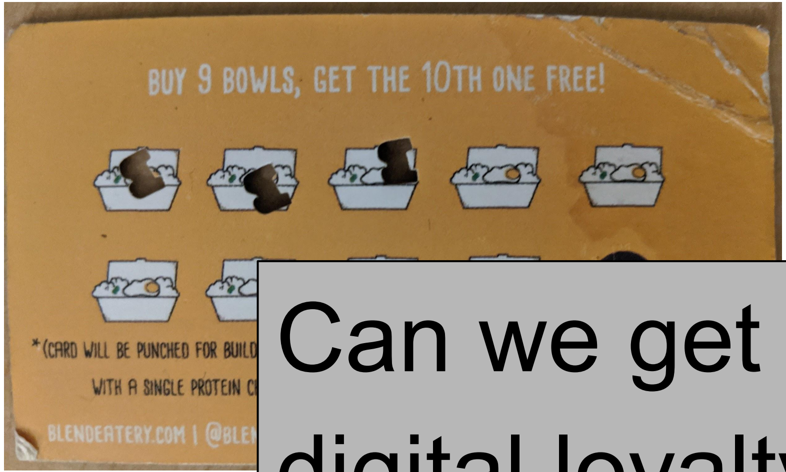
Customer convenience
No lost cards
Better bookkeeping
Hard to Counterfeit
Contactless

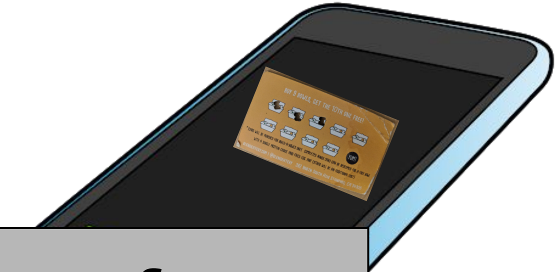

<u>Why go digital?</u>

Customer convenience
No lost cards
Better bookkeeping
Hard to Counterfeit
Contactless

<u>Why NOT go digital?</u>

Digital loyalty programs can be data-stealing monsters*



# Can we get the benefits of digital loyalty programs without sacrificing privacy?

Why go dig

Customer c ... ns can be

No lost car ... s*

Better bookkeeping

Hard to Counterfeit

Contactless

# Existing Approaches

Anonymous credentials, eCash, uCentive

- Give customers an unlinkable token for each purchase
- Customer redeems by presenting a bunch of tokens
- Work scales linearly in the number of hole punches

# Existing Approaches

Recent line of work: BBA [JR16], BBA+ [HHNR17], UACS [BBDE19], Bobolz et al.[BEKS20]

- "Black-box accumulation"/"Updatable anonymous credentials"
- Punch card storage and performance independent of number of punches
- Support for broader functionalities
  - e.g., Offline double spending, negative points, partial redemption
- Performance could be improved -- reliance on pairings, involved proofs
- Mismatch between scheme and punch card deployment scenario

# Our Work

Focus on real requirements for punch cards:

- No long-term user identity tied to a public key
- No server work to issue cards (avoids DoS)
- Minimizes round complexity

# Our Work

Focus on real requirements for punch cards:

- No long-term user identity tied to a public key
- No server work to issue cards (avoids DoS)
- Minimizes round complexity

Improves performance:

- Removes reliance on pairings
- 14x faster card punch, 25x less communication
- 394x faster card redemption, 62x less communication

# Punch Card Functionality

Server setup: initialize server secrets, database of redeemed cards

Card issuance: issue a fresh punch card

Card punch: add a punch to an existing punch card

Card redemption/validation: submit a completed punch card for a reward

# Punch Card Security

**Privacy:** Server can’t link any issuances, punches, or redemptions to each other

Server can simulate everything it sees when issuing/punching/redeeming a card.

**Soundness:** Client can’t redeem more punches than it has received

Challenger allows adversary to punch and redeem cards. Adversary wins if more punches redeemed than given.

# First Attempt

Idea: server raises group element to secret power for each punch

# First Attempt

Client

Issue

$p_0 \leftarrow_R G$

Server

Setup

$sk \leftarrow_R Z_q$

# First Attempt

Client

Issue

$p_0 \leftarrow_R G$

Punch

Server

Setup

$sk \leftarrow_R Z_q$

Punch

$p_i$ →

← $p_{i+1}$

$p_{i+1} \leftarrow p_i^{sk}$

# First Attempt

Client

<u>Issue</u>

$p_0 \leftarrow_R G$

<u>Punch</u>

$p_i$ →

← $p_{i+1}$

<u>Redeem</u>

$p_0, p_n$ →

Server

<u>Setup</u>

$sk \leftarrow_R Z_q$

<u>Punch</u>

$p_{i+1} \leftarrow p_i^{sk}$

<u>Verify</u>

Accept iff 1. $p_n = p_0^{sk\char`^n}$

2. $p_0$ not redeemed before

# First Attempt

Client

Issue

$p_0 \leftarrow_R G$

Punch

Redeem

**Neither private nor sound!**

$p_i$ →

← $p_{i+1}$

$p_0, p_n$ →

Server

Setup

$sk \leftarrow_R Z_q$

Punch

$p_{i+1} \leftarrow p_i^{sk}$

Verify

Accept iff 1. $p_n = p_0^{sk\wedge n}$

2. $p_0$ not redeemed before

# Adding Privacy

Idea: client masks punch card before sending to server

# Adding Privacy

Idea: client masks punch card before sending to server

Client

Server

<u>Punch</u>

<u>Punch</u>

$m \leftarrow_R Z_q$

$p_i' \leftarrow p_i^m$

$p_i'$ →

$p'_{i+1} \leftarrow (p'_i)^{sk}$

← $p'_{i+1}$

$p_{i+1} \leftarrow (p'_{i+1})^{m^{-1}}$

# Adding Privacy

Idea: client masks punch card before sending to server

Client

Server

Punch

$m \leftarrow_R Z_q$

$p_i' \leftarrow p_i^m$

$p_i'$ →

Punch

$p'_{i+1} \leftarrow (p'_i)^{sk}$

← $p'_{i+1}$

$p_{i+1} \leftarrow (p'_{i+1})^{m\text{^-}1}$

Only semihonest security!

# Malicious Privacy

Malicious attack: server raises one punch card to a different power


# Malicious Privacy

Malicious attack: server raises one punch card to a different power



Defense: Server provides proof that it raised card to the same power each time

# Malicious Privacy

Malicious attack: server raises one punch card to a different power



Defense: Server provides proof that it raised card to the same power each time

Modify server setup to include $pk = g^{sk}$

Use Chaum-Pedersen Proof: Given $g, pk, p$,

prove knowledge of $sk$ s.t. $pk=g^{sk}$, $p'=p^{sk}$

# Adding Soundness

Current redeem process: client sends $p_0, p_n$

Server checks $p_n = (p_0)^{sk\wedge n}$, $p_0$ not redeemed before

# Adding Soundness

Current redeem process: client sends $p_0, p_n$

Server checks $p_n = (p_0)^{sk\wedge n}$, $p_0$ not redeemed before

Attack:

1. Malicious client sends $p_0, p_n$
2. Server checks $p_n = (p_0)^{sk\wedge n}$, $p_0$ not redeemed before, redeems *n* points

# Adding Soundness

Current redeem process: client sends $p_0, p_n$

Server checks $p_n = (p_0)^{sk\text{^}n}$, $p_0$ not redeemed before

<u>Attack:</u>

1. Malicious client sends $p_0, p_n$
2. Server checks $p_n = (p_0)^{sk\text{^}n}$, $p_0$ not redeemed before, redeems *n* points
3. Malicious client gets another punch on $p_n$, acquires $p_{n+1}$

# Adding Soundness

Current redeem process: client sends $p_0, p_n$

Server checks $p_n = (p_0)^{sk\char`^n}$, $p_0$ not redeemed before

Attack:

1. Malicious client sends $p_0, p_n$
2. Server checks $p_n = (p_0)^{sk\char`^n}$, $p_0$ not redeemed before, redeems *n* points
3. Malicious client gets another punch on $p_n$, acquires $p_{n+1}$
4. Malicious client sends $p_1$, $p_{n+1}$
5. Server checks $p_{n+1} = (p_1)^{sk\char`^(n+1)}$, $p_1$ not redeemed before, redeems *n* points

# Adding Soundness

Current redeem process: client sends $p_0, p_n$

Server checks $p_n = (p_0)^{sk^n}$, $p_0$ not redeemed before

Attack:

1. Malicious client sends $p_0, p_n$
2. Server checks $p_n = (p_0)^{sk^n}$, $p_0$ not redeemed before, redeems *n* points
3. Malicious client gets another punch on $p_n$, acquires $p_{n+1}$
4. Malicious client sends $p_1$, $p_{n+1}$
5. Server checks $p_{n+1} = (p_1)^{sk^{(n+1)}}$, $p_1$ not redeemed before, redeems *n* points

Client gets n+1 punches, redeems 2n points, breaks soundness

# Adding Soundness

Idea: client can't redeem a punch card $p_0$ unless it knows the preimage of a hash function (modeled as RO) that outputs $p_0$

Client

Server

# Adding Soundness

Idea: client can't redeem a punch card $p_0$ unless it knows the preimage of a hash function (modeled as RO) that outputs $p_0$

Client

Issue

$id \leftarrow_R \{0,1\}^\lambda$

$p_0 \leftarrow H(id)$

Server

Setup

$sk \leftarrow_R Z_q$

$pk \leftarrow_R g^{sk}$

# Adding Soundness

Idea: client can't redeem a punch card $p_0$ unless it knows the preimage of a hash function (modeled as RO) that outputs $p_0$

Client

<u>Issue</u>

$id \leftarrow_R \{0,1\}^\lambda$

$p_0 \leftarrow H(id)$

...

<u>Redeem</u>

$id, p_n$ →

Server

<u>Setup</u>

$sk \leftarrow_R Z_q$

$pk \leftarrow_R g^{sk}$

...

<u>Verify</u>

Accept iff 1. $p_n = H(id)^{sk\text{^}n}$

2. *id* not redeemed before

# Proving Soundness

Proof in Algebraic Group Model (most prior work proven in more restrictive GGM)

Adversaries in the AGM must accompany each group element they send with a representation of that group element in terms of previously seen elements

In this model, DDH-style assumptions are equivalent to discrete log

# Proving Soundness

Proof in Algebraic Group Model (most prior work proven in more restrictive GGM)

Adversaries in the AGM must accompany each group element they send with a representation of that group element in terms of previously seen elements

In this model, DDH-style assumptions are equivalent to discrete log

Proof relies on hardness of *d*-discrete log assumption: given $g, g^x, g^{x^2}, g^{x^d}$, find $x$.

# Proving Soundness

1. Let $d$-dlog group elements be $X_0 = g$, $X_1 = g^x$, …, $X_d = g^{x^d}$

# Proving Soundness

1. Let *d*-dlog group elements be $X_0=g$, $X_1= g^x$, …, $X_d=g^{x^d}$
2. Set the server secret to be the *d*-dlog solution $x$

# Proving Soundness

1. Let $d$-dlog group elements be $X_0=g$, $X_1= g^x$, …, $X_d=g^{x^d}$
2. Set the server secret to be the $d$-dlog solution $x$
3. Program RO so that every hash output is of the form $g^r$ where the soundness challenger knows $r \leftarrow_R Z_q$ (but output still looks random to the adversary)

# Proving Soundness

1. Let *d*-dlog group elements be $X_0=g$, $X_1= g^x$, …, $X_d=g^{x\wedge d}$
2. Set the server secret to be the *d*-dlog solution $x$
3. Program RO so that every hash output is of the form $g^r$ where the soundness challenger knows $r \leftarrow_R Z_q$ (but output still looks random to the adversary)
4. To punch a card, look at algebraic representation of punch card and replace each $X_i$ with $X_{i+1}$.

# Proving Soundness

1. Let *d*-dlog group elements be $X_0=g, X_1= g^x, \ldots, X_d=g^{x\wedge d}$
2. Set the server secret to be the *d*-dlog solution $x$
3. Program RO so that every hash output is of the form $g^r$ where the soundness challenger knows $r \leftarrow_R Z_q$ (but output still looks random to the adversary)
4. To punch a card, look at algebraic representation of punch card and replace each $X_i$ with $X_{i+1}$.
5. Soundness adversary who wins the soundness game produces a punch card whose representation does not include $X_n^r$, which gives the challenger 2 representations of $X_n$. It can use this to break discrete log.

# Implementation

Java (Android) wrapper around Rust implementation

Main construction implemented using curve25519-dalek

Evaluated on Pixel 1 (client) and recent Thinkpad laptop with i5 processor (server)

# Implementation

Java (Android) wrapper around Rust implementation

Main construction implemented using curve25519-dalek

Evaluated on Pixel 1 (client) and recent Thinkpad laptop with i5 processor (server)

Evaluated on empty database of used cards and database of 1,000,000 used cards, numbers comparable (in prior work, larger DB is much more expensive)

# Implementation

Java (Android) wrapper around Rust implementation

Main construction implemented using curve25519-dalek

Evaluated on Pixel 1 (client) and recent Thinkpad laptop with i5 processor (server)

Evaluated on empty database of used cards and database of 1,000,000 used cards, numbers comparable (in prior work, larger DB is much more expensive)

| | ServerSetup | Issue | ServerPunch | ClientPunch | ClientRedeem | ServerVerify |
|---|---|---|---|---|---|---|
| Computation Time (ms) | 0.019 | 0.304 | 0.134 | 4.314 | 0.890 | 0.064 |
| Data Sent (Bytes) | 32 | 0 | 128 | 32 | 64 | 0 |

# Computation Comparison

Prior work evaluated on comparable hardware (Pixel/OnePlus 3, i7 processor)

Prior work uses BN curves with slightly lower security (~100 bits)

Each prior work dominated others in one part of protocol, our work improves on the best prior work in each category by order(s) of magnitude

| | Issuing a Card | Punching a Card | Redeeming a Card |
|---|---|---|---|
| BBA+ scheme | 115.27 | 385.61 | 375.73 |
| UACS scheme | 86 | 127 | 454 |
| Bobolz et al. scheme | 130 | 64 | 1254 |
| Our main scheme | 0.304 (282.99× faster) | 4.448 (14.4× faster) | 0.954 (393.8× faster) |

(All times in ms)

# Communication Comparison

Only one prior work reports communication costs

Our scheme requires no server involvement to issue a card

| | Issuing a Card | Punching a Card | Redeeming a Card |
|---|---|---|---|
| BBA+ scheme | 992 | 4048 | 3984 |
| Our main scheme | 0 | 160 (25.3× reduction) | 64 (62.3× reduction) |

(All sizes in bytes)

# Fast Privacy-Preserving Punch Cards

Key takeaways:

- 14x faster card punch, 25x less communication than prior work
- 394x faster card redemption, 62x less communication than prior work
- Qualitative improvements to better capture punch card setting

See paper for more details and extensions

Paper: `https://arxiv.org/pdf/2006.06079.pdf`

Code: `https://github.com/SabaEskandarian/PunchCard`

Contact: saba@cs.stanford.edu