

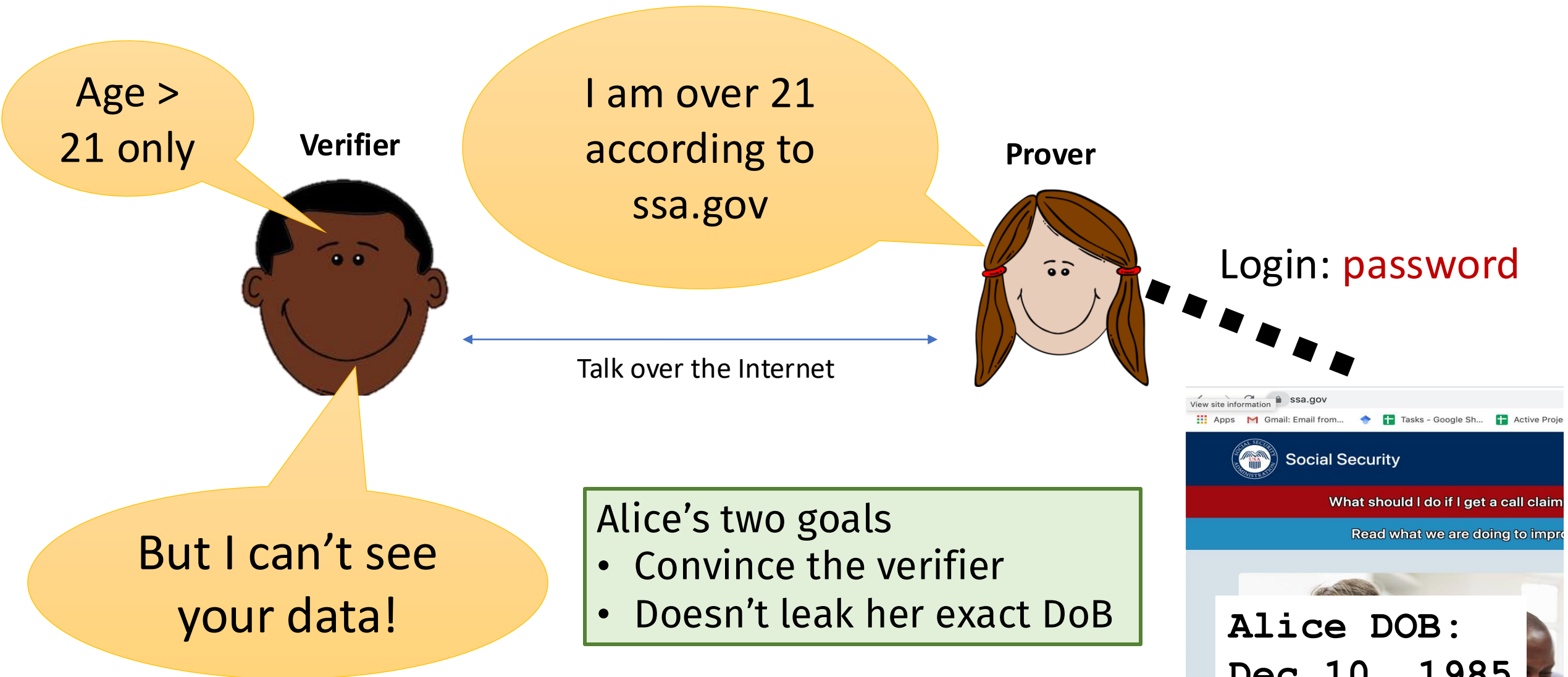
# Oracles, “zkTLS”, and Data Liberation

Fan Zhang  
Yale University & IC3

Ethereum Research Funding Forum

# How to prove that you have seen a particular web page?

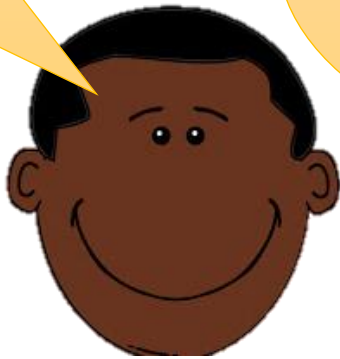
# Example #1: Age Verification



# Example #2: (Decentralized) Finance

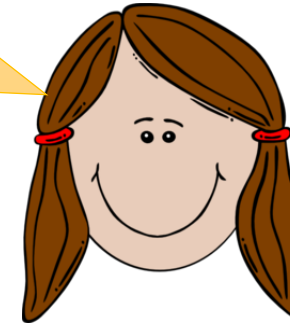
FICO >  
700 only

Verifier



My credit score is  
> 700 according to  
Discover.com

Prover



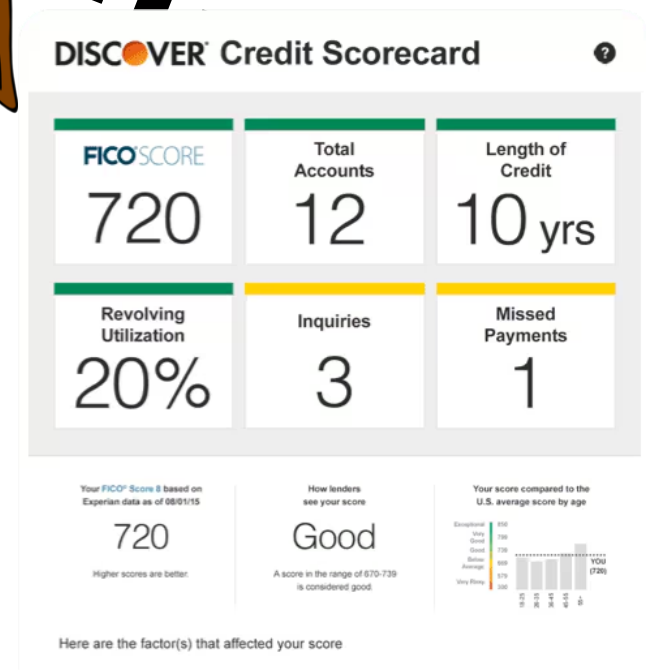
Login: password

E.g., Lender (or a lending smart contract)

Alice's two goals

- Convince the verifier
- Doesn't leak other information (e.g., account balance)

ETH NYC 2025



# Example #3: Employment status

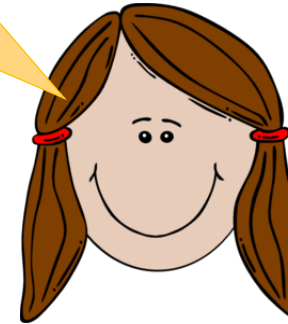
Verifier



E.g., Mortgage approver

I'm employed by  
Yale according to  
workday.yale.edu

Prover



Login: password

- Alice's two goals
- Convince the verifier
  - Doesn't leak other information (e.g., her salary)

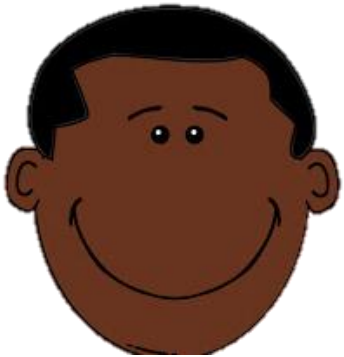


# Goal: Proving *statements* about TLS-protected data

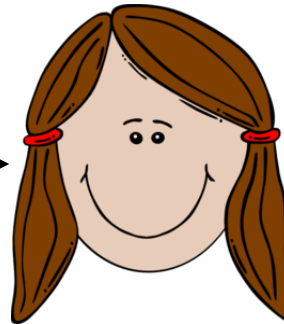
TLS (Transport Layer Security): the security protocol encrypting the web.

My DoB (Y/M/D) on SSA.gov satisfies that  $2022 - Y > 18$ .

Verifier



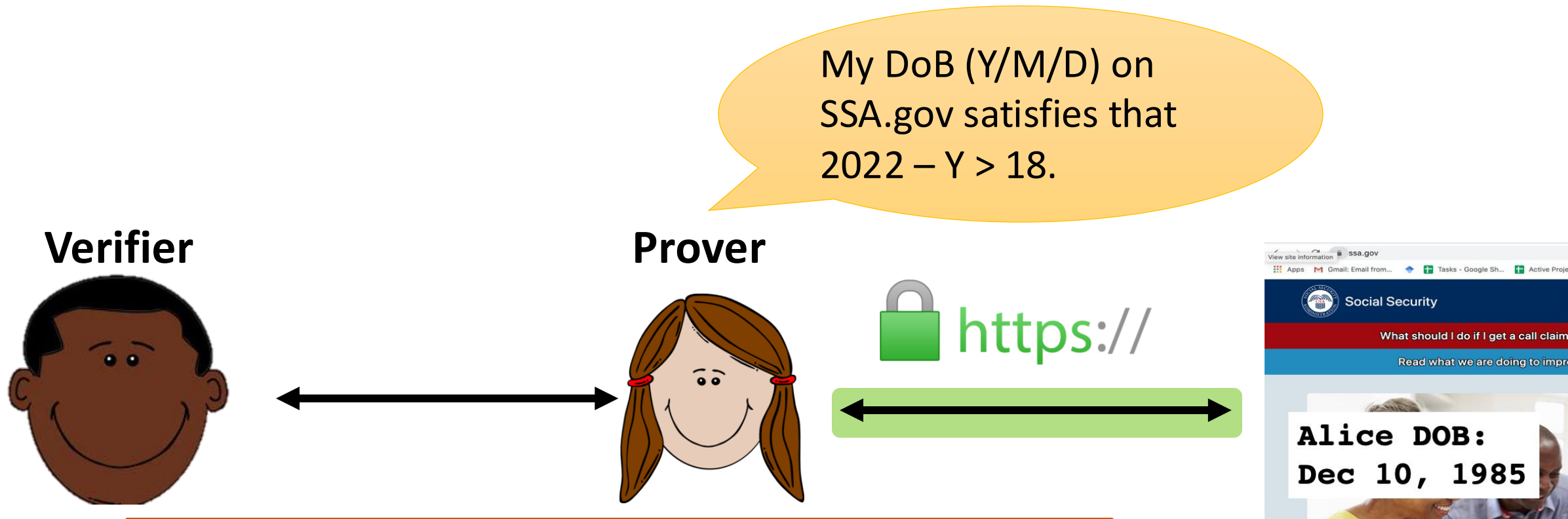
Prover



Two goals

- **Integrity:** Prover can't fool the verifier.
- **Privacy:** Verifier doesn't learn more than the statement being true.

# Goal: Proving *statements* about TLS-protected data

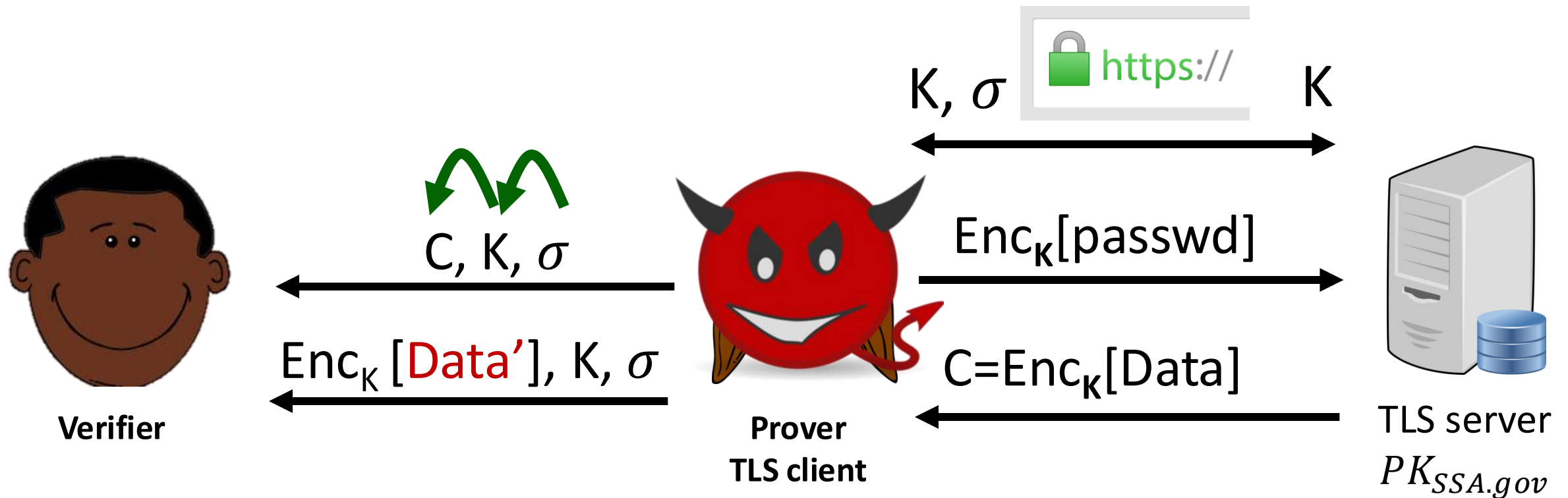


Let's rule out some bad ideas:

#1: Prover to send over the SSA.gov password

#2: Prover to send over a screenshot of SSA.gov

# Problem: TLS doesn't support provenance



**TLS doesn't sign the data!**  
Pro: non-repudiation / deniability  
Con: no provenance.



# How about we just change the web?

- E.g., Change TLS to sign the data (TLS-N) or add signature to HTTP messages (e.g., RFC-9421)
- Challenges: adoption barrier
  - making deniability impossible

Ritzdorf, Hubert, et al. "TLS-N: Non-repudiation over TLS Enabling Ubiquitous Content Signing." In *NDSS*, 2018.

RFC-9421

HTTP Message Signatures

## Abstract

This document describes a mechanism for creating, encoding, and verifying digital signatures or message authentication codes over components of an HTTP message. This mechanism supports use cases where the full HTTP message may not be known to the signer and where the message may be transformed (e.g., by intermediaries) before reaching the verifier. This document also describes a means for requesting that a signature be applied to a subsequent HTTP message in an ongoing HTTP exchange.

# Our solutions: Town Crier and DECO

- Unique feature: requires no changes to websites

 SGX



Use **TEEs**

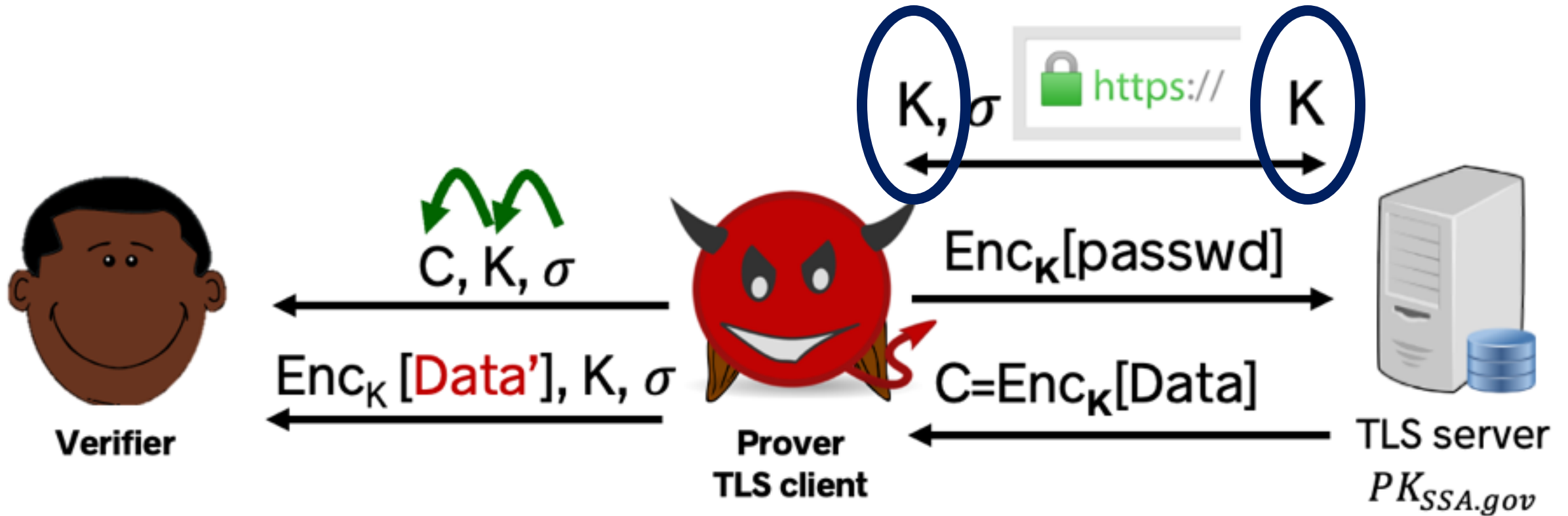


Use **cryptographic  
protocols**

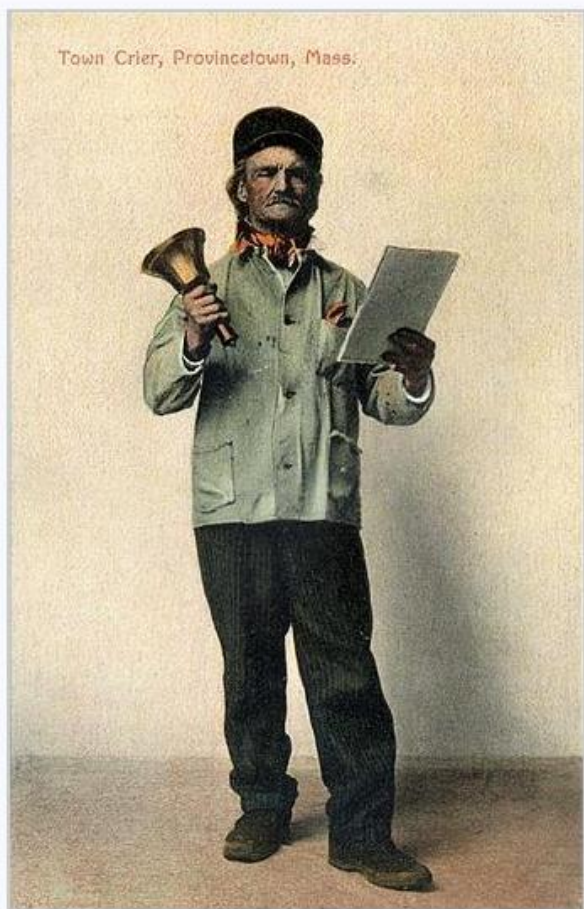
# Now known as “zkTLS” protocols

- Town Crier
- “DECO” regime
  - DIDO: Data Provenance from Restricted TLS 1.3 Websites
  - Janus: Fast Privacy-Preserving Data Provenance for TLS
  - Lightweight Authentication of Web Data via Garble-Then-Prove
  - ORIGO: Proving Provenance of Sensitive Data with Constant Communication
  - DiStefano: Decentralized Infrastructure for Sharing Trusted Encrypted Facts and Nothing More

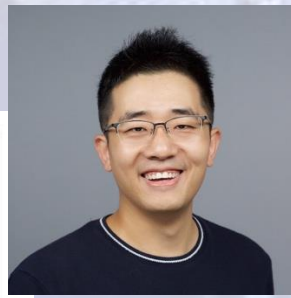
# Problem: TLS doesn't support provenance



Our approach: hide session key from the prover until she commits.



Town crier of  
Provincetown,  
Massachusetts, in 1909



# Town Crier: An Authenticated Data Feed for Smart Contracts

F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town Crier: An Authenticated Data Feed for Smart Contracts," in ACM CCS '16.

[https://en.wikipedia.org/wiki/Town\\_crier](https://en.wikipedia.org/wiki/Town_crier)



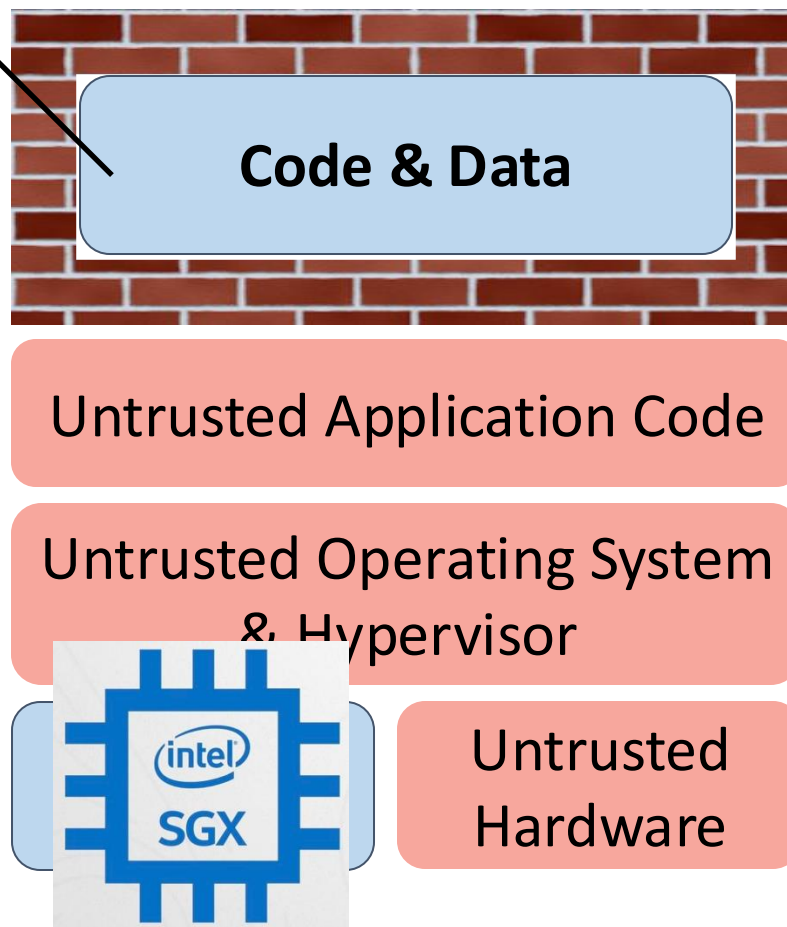
# TEE: isolated execution

## Integrity



Other software and **even OS** cannot tamper with control flow.

“Enclave”

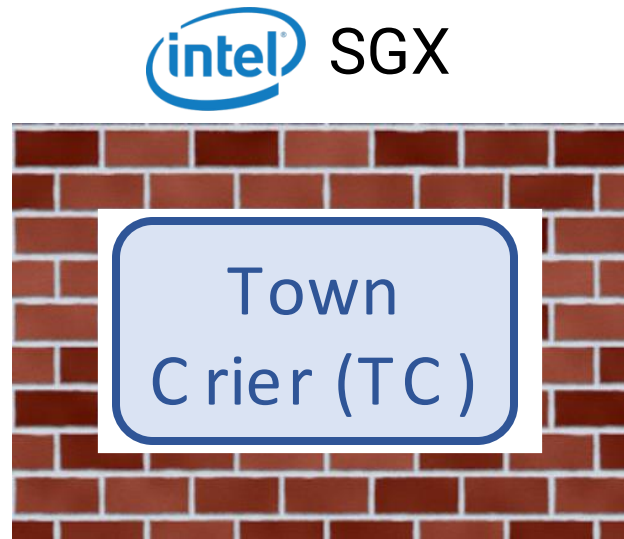


## Confidentiality



Other software and **even OS** can learn nothing about the internal state.

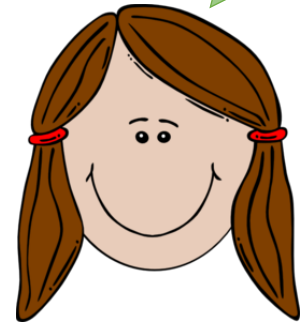
# Trusted hardware: Remote attestation



attestation att =  
 $\Sigma_{\text{SGX}}[ \text{Build(TC)} \parallel \text{Data} ]$



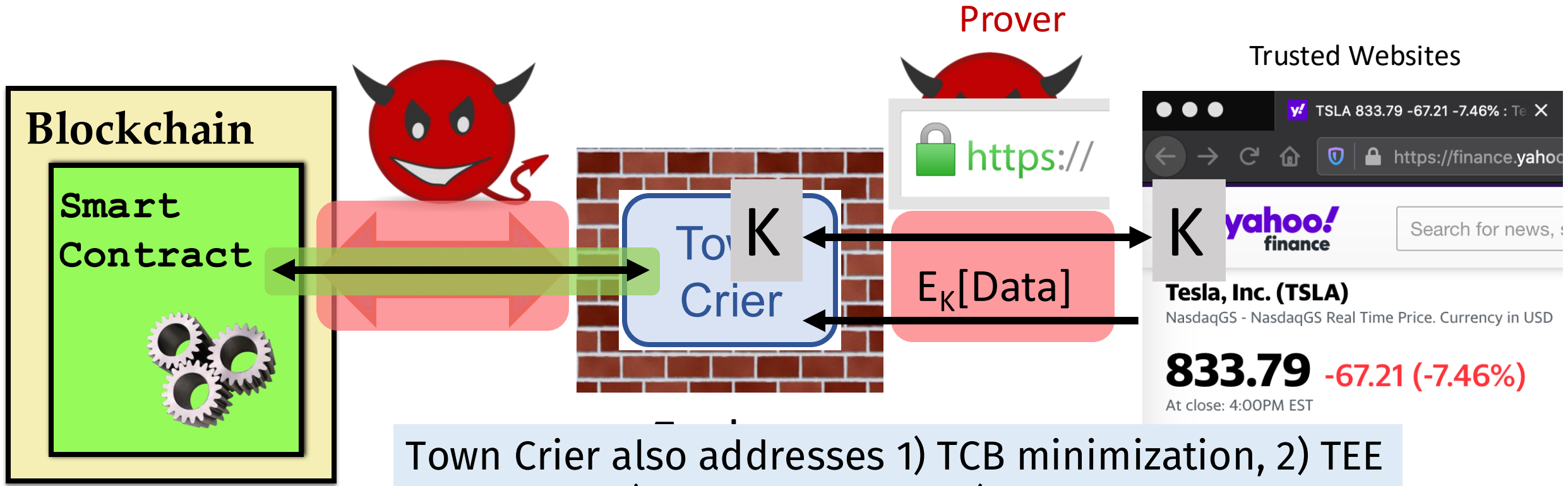
It's indeed Town Crier (TC) running in a genuine Intel SGX.



Remote entity

# Town Crier

- Main idea: Terminates TLS in TEE
- Using attestation to prove authenticity



Town Crier also addresses 1) TCB minimization, 2) TEE formalism, 3) gas optimization (e.g., avoid verifying attestations on-chain), 4) gas neutrality, etc.



# DECO (CCS'21)

- Can we do the same without TEEs?
  - An interesting challenge
  - TEEs are vulnerable to side channel attacks
- DECO
  - Requires **no trusted hardware**
  - Requires **no server-side modifications**
  - Works with modern TLS versions (1.2 & 1.3)
- Not without compromise (designated-verifier)

# DECO: Liberating Web Data Using Decentralized Oracles for TLS

The extended version. Updated on August 6, 2024.

Fan Zhang\*  
Cornell Tech

Deepak Maram\*  
Cornell Tech

Harjasleen Malvai\*  
Cornell University

Steven Goldfeder\*  
Cornell Tech

Ari Juels\*  
Cornell Tech

## ABSTRACT

Thanks to the widespread deployment of TLS, users can access private data over channels with end-to-end confidentiality and integrity. What they cannot do, however, is prove to third parties the *provenance* of such data, i.e., that it genuinely came from a particular website. Existing approaches either introduce undesirable trust assumptions or require server-side modifications.

Users' private data is thus locked up at its point of origin. Users cannot export data in an integrity-protected way to other applications without help and permission from the current data holder.

We propose DECO (short for decentralized oracle) to address the above problems. DECO allows users to prove that a piece of data accessed via TLS came from a particular website and optionally prove statements about such data in zero-knowledge, keeping the data itself secret. DECO is the first such system that works without trusted hardware or server-side modifications.

DECO can liberate private data from centralized web-service silos, making it accessible to a rich spectrum of applications. To demonstrate the power of DECO, we implement three applications that are hard to achieve without it: a private financial instrument using smart contracts, converting legacy credentials to anonymous credentials, and verifiable claims against price discrimination.

Specifically, when a user accesses data online via TLS, she cannot securely *export* it, without help (hence permission) from the current data holder. Vast quantities of private data are thus intentionally or unintentionally locked up in the “deep web”—the part of the web that isn’t publicly accessible.

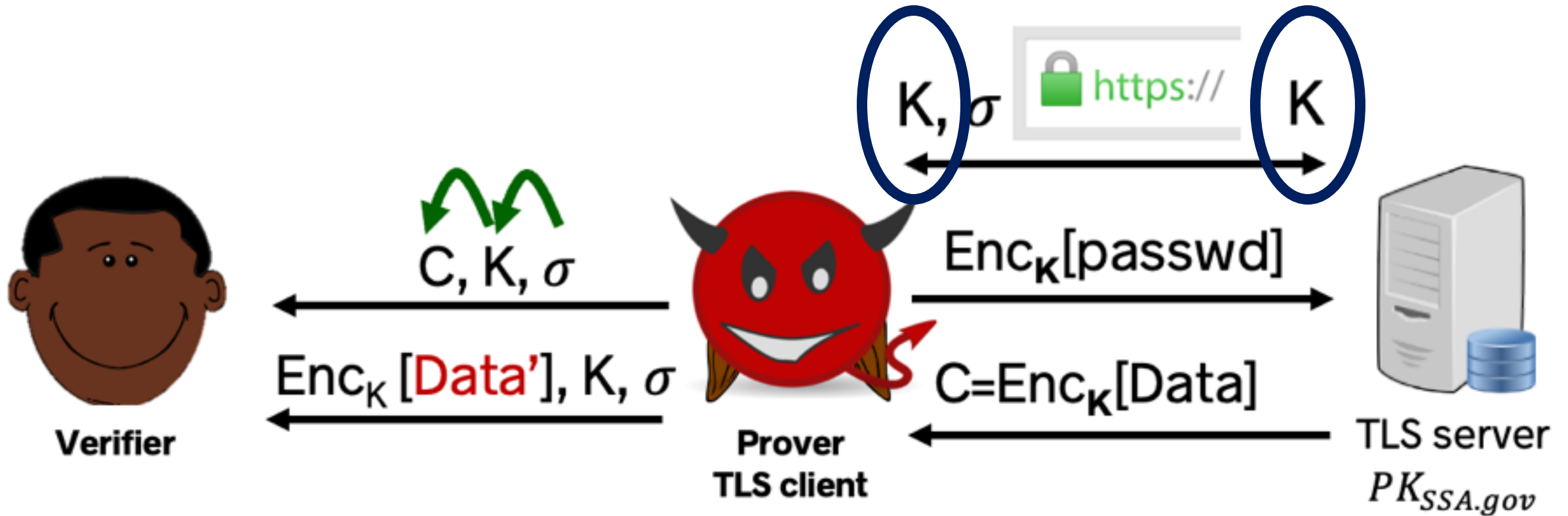
To understand the problem, suppose Alice wants to prove to Bob that she’s over 18. Currently, age verification services [1] require users to upload IDs and detailed personal information, which raises privacy concerns. But various websites, such as company payroll records or DMV websites, in principle store and serve verified birth dates. Alice could send a screenshot of her birth date from such a site, but this is easily forged. And even if the screenshot could somehow be proven authentic, it would leak information—revealing her exact birth date, not just that she’s over 18.

Proposed to prove provenance of online data to smart contracts, *oracles* are a step towards exporting TLS-protected data to other systems with provenance and integrity assurances. Existing schemes, however, have serious limitations. They either only work with deprecated TLS versions and offer no privacy from the oracle (e.g., TL-SNotary [7]) or rely on trusted hardware (e.g., Town Crier [78]), against which various attacks have recently emerged, e.g., [24].

Another class of oracle schemes assumes server-side cooperation at service providers (e.g., [24]).

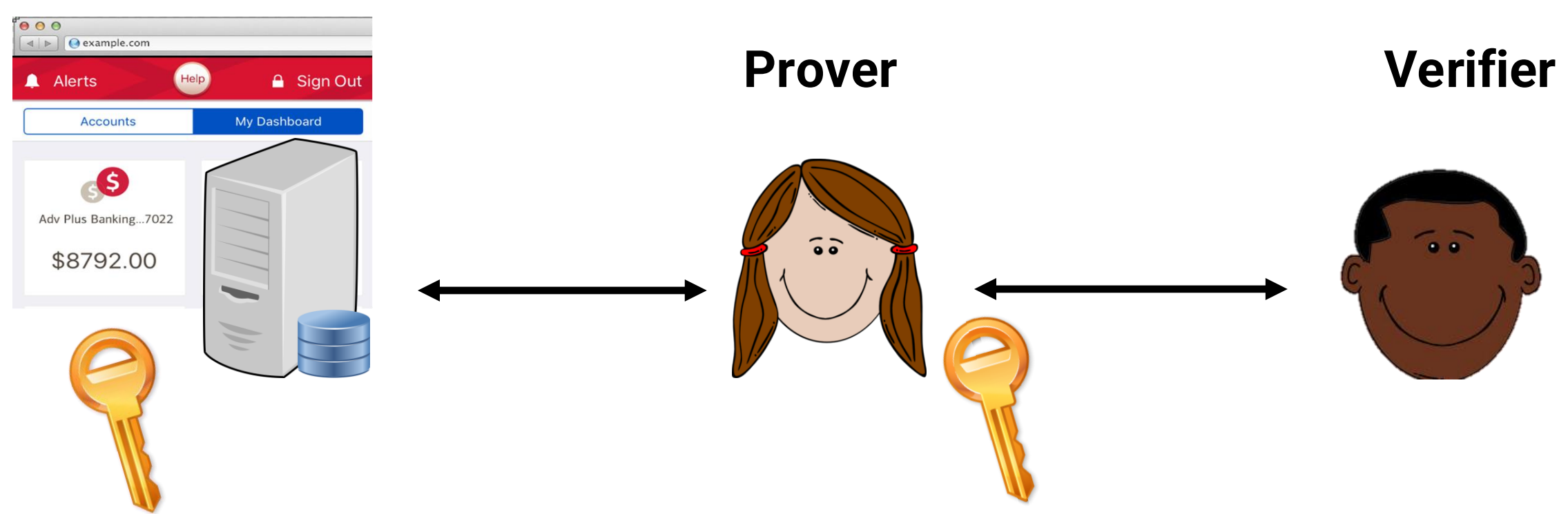


# Problem: TLS doesn't support provenance

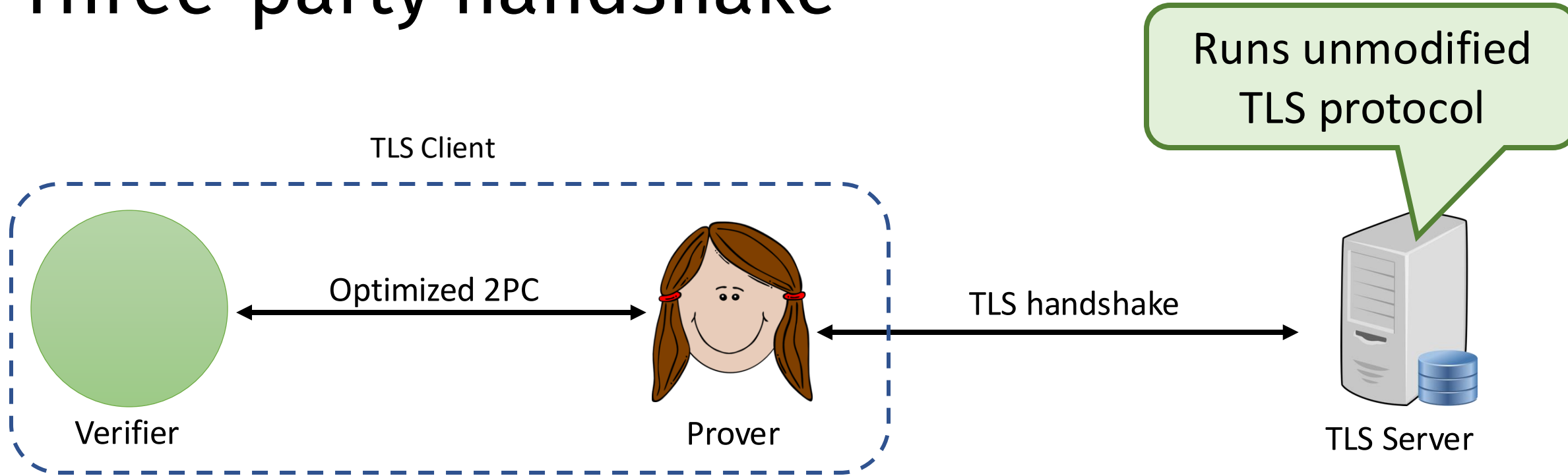


Can we hide session key from the prover without TEEs?

# Prevent cheating by **splitting session keys**



# Three-party handshake



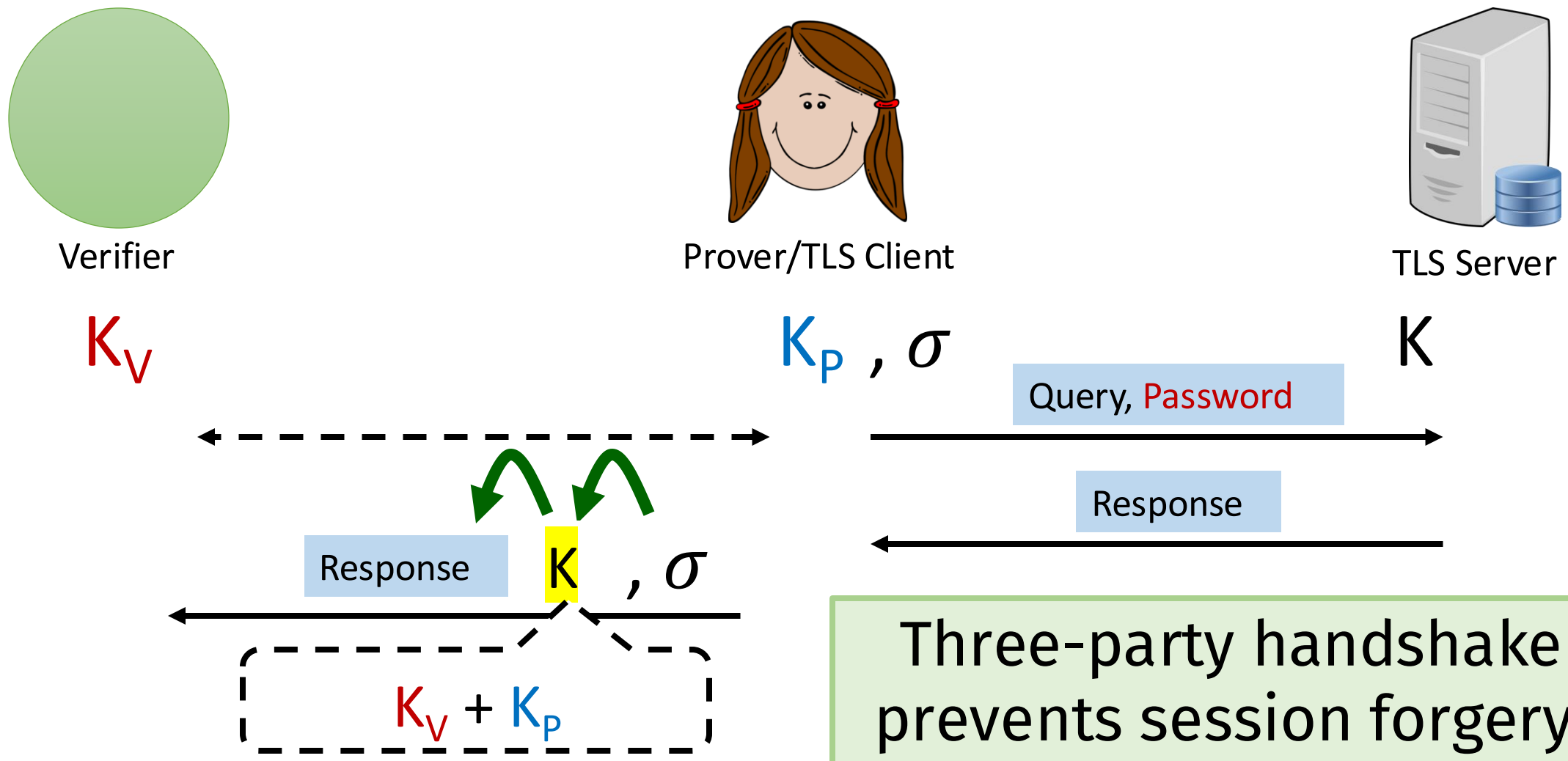
## Main technical challenges

- Securely splitting client logic in TLS handshake in 2PC
- Optimizing 2PC performance



This denotes a TLS ciphertext.

# After the three-party handshake

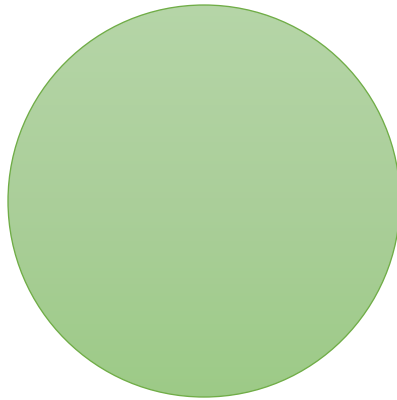


Three-party handshake prevents session forgery.

# Commit then prove

Convinced!

Verifier



Prover



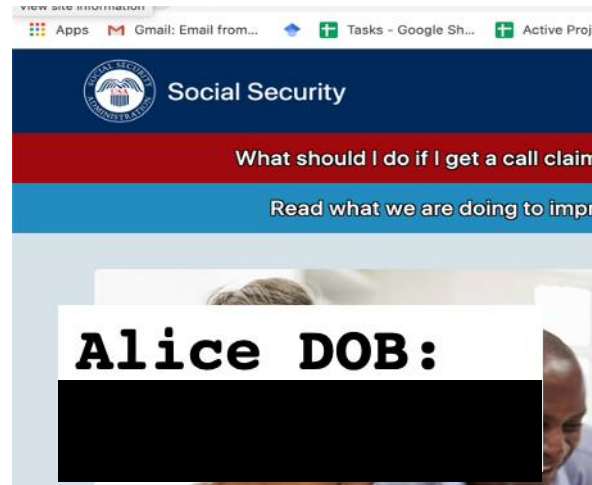
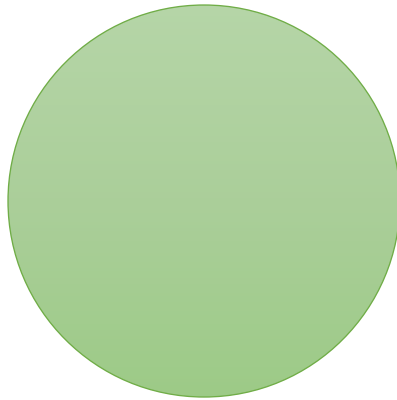
I'm over 18.

**Query privacy:** verifier never sees the **password**.

# Commit then prove

Convinced!

Verifier



←  
A proof that "2022 -  
DOB.Year > 18"

Prover



I'm over 18.

"zkTLS"

**Fine-grain privacy** with zero-knowledge proofs





1. Commit

Terminating TLS  
sessions in TEEs

Three-party HS



2. Prove

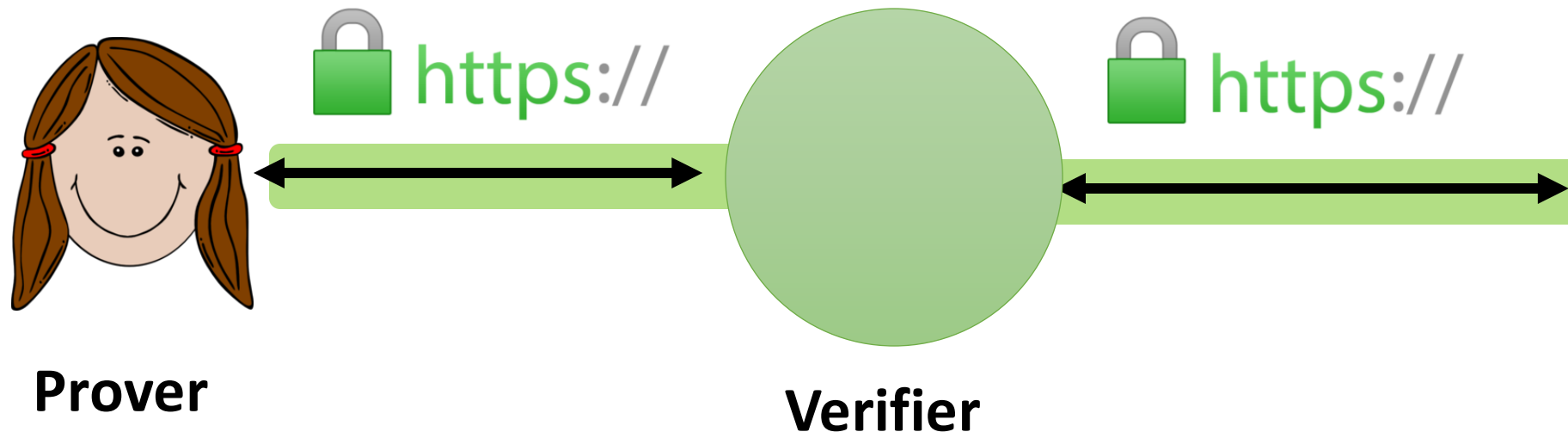
TEE attestations

Zero-knowledge  
proofs

Zero-knowledge  
proofs

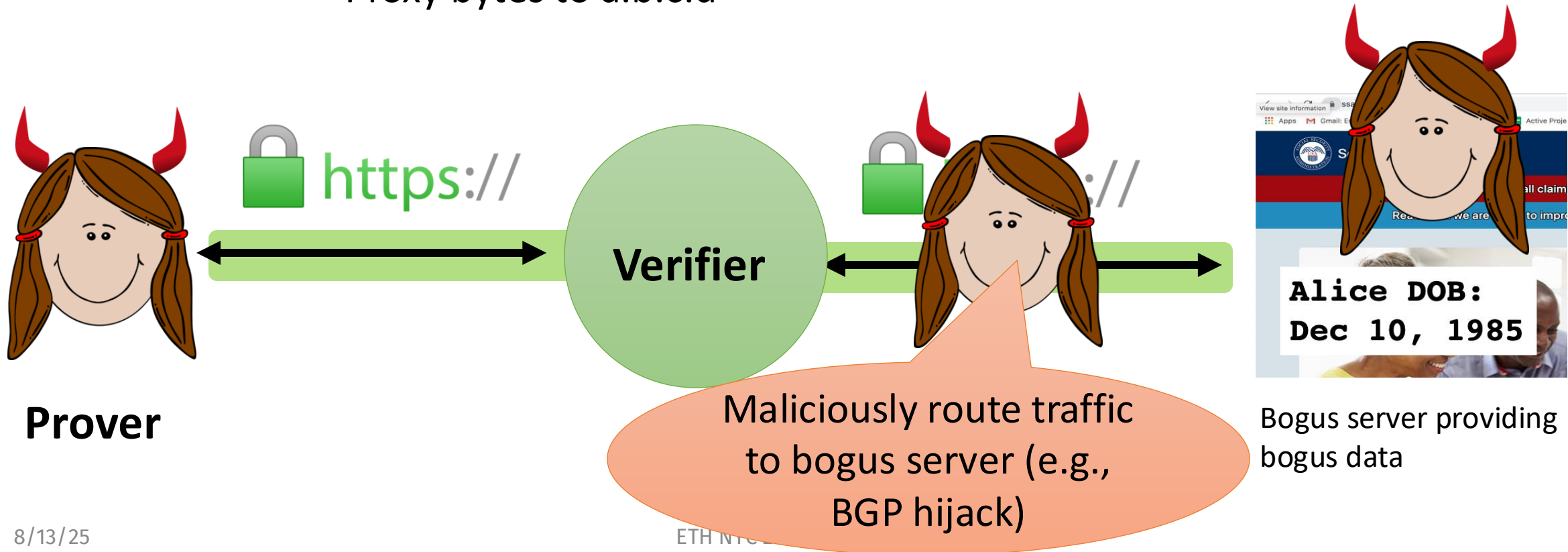
# Proxy Mode: alternative to TPHS

- Idea: proxy TLS traffic through Verifier
- Hope: the **recorded traffic** is a **commitment** to the plaintext
  - No need for special handshake
  - Efficient alternative to third-party handshake



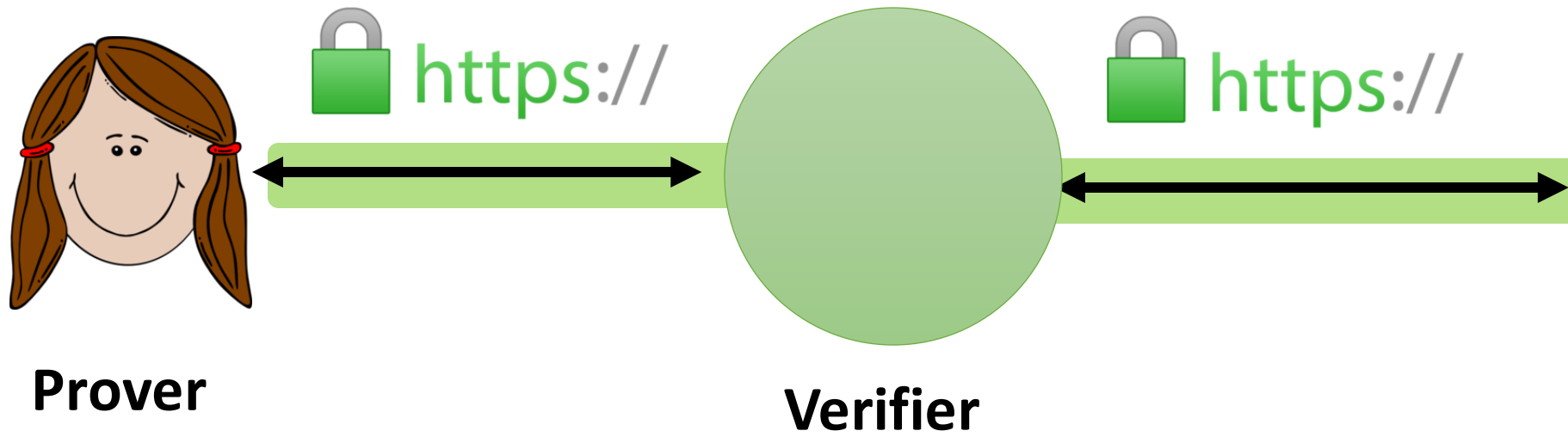
# Problem #1: Relying on IP for authenticity

- Query DNS ssa.gov -> a.b.c.d (an IP addr)
- Proxy bytes to a.b.c.d



# Problem #2: Does **recorded traffic** commit to the plaintext?

- Suppose the traffic encrypts message  $m$  using session key  $K$ .
- Can prover forge  $K'$  that also successfully decrypt the recorded traffic to  $m$ '?



# Encryption is not necessarily committing

- Ciphertext  $c = \text{Enc}(K, m)$  may not commit to  $(K, m)$
- I.e., one can find  $(K', m')$  such that  $\text{Enc}(K', m') = c$
- For instances
  - CBC-HMAC commits when keys are derived from PRF
  - AEAD ciphers does not commit in general

# How to deal with AEAD?

- Our solution in DECO: Add key binding proofs
- Luo et al: Proxy is enough under practical assumptions
  - E.g., HTTP headers can serve as an invariant

## **Proxying is Enough: Security of Proxying in TLS Oracles and AEAD Context Unforgeability**

Zhongtang Luo  
*Purdue University*  
*luo401@purdue.edu*

Yanxue Jia  
*Purdue University*  
*jia168@purdue.edu*

Yaobin Shen  
*Xiamen University*  
*yaobin.shen@xmu.edu.cn*

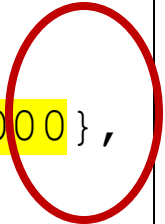
Aniket Kate  
*Purdue University / Supra Research*  
*aniket@purdue.edu*

# Other Technical Subtleties

- **Context integrity** problem with privacy
  - Attackers may exploit the privacy to alter the meaning
  - Solution: careful reasoning based on the formal grammar

```
{  
  "account": "alice",  
  "SSN": 1234567890,  
  "balance": 100,  
  "last_month": {"balance": 8000},  
}
```

```
{  
  "account": "alice",  
  "SSN": [REDACTED]  
  [REDACTED] "balance": 8000},  
}
```

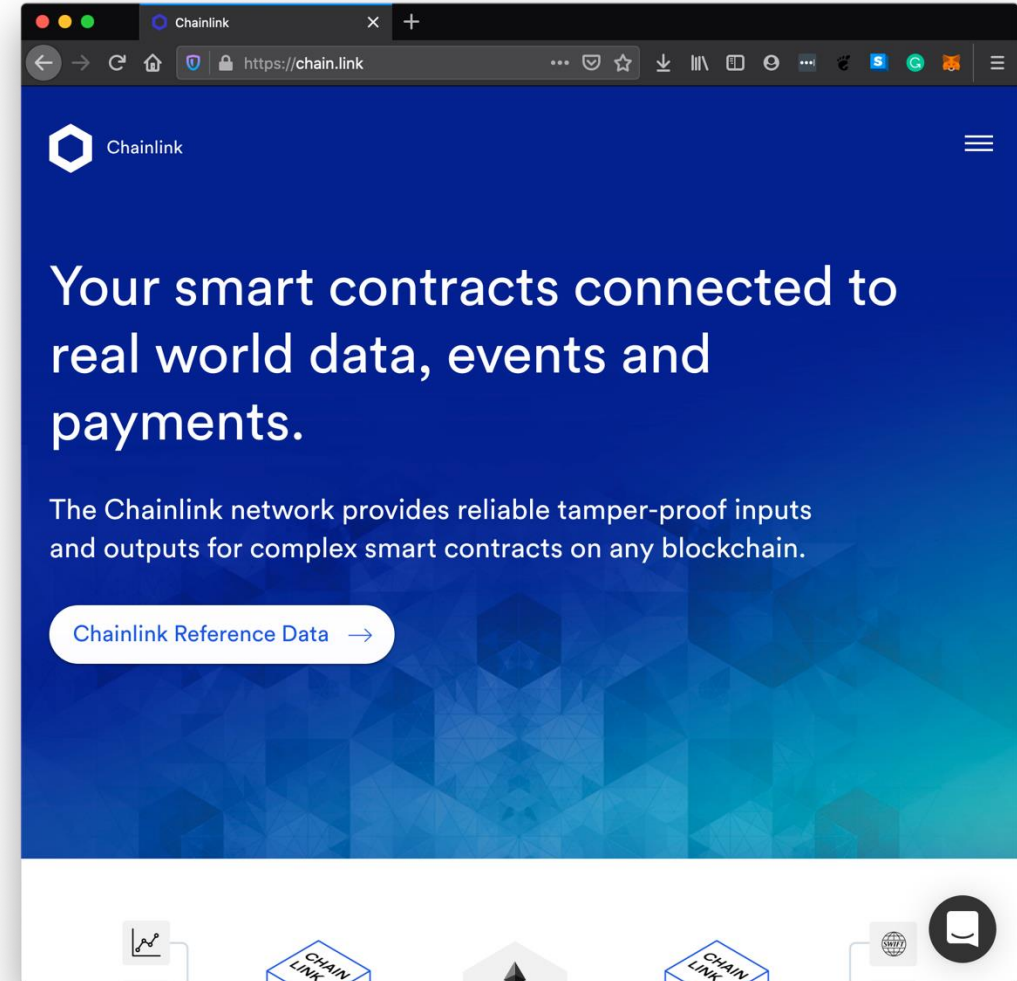


DECO addressed this problem in some special cases.  
Malvai *et al.* did a thorough investigation  
(<https://eprint.iacr.org/2024/562>).

# DECO and Town Crier acquired by Chainlink



https://chain.link





# DECO Sandbox

Experience the power of zero-knowledge proofs and privacy-preserving data verification for onchain finance by leveraging pre-configured use cases or creating your own.

Start exploring



## Optimize User Onboarding and Streamline Operations

Test DECO's streamlined data verification processes to enhance onboarding speed and reduce costs by minimizing duplicate checks and manual processes.



## Strengthen Compliance Without Data Exposure

Explore how DECO's privacy-preserving data verification strengthens compliance by improving data protection while helping users meet certain regulatory requirements.



## Drive Privacy Innovation in Onchain Finance

Trial how DECO's oracles and zero-knowledge proofs enable cutting-edge privacy-first solutions, opening up new use cases in onchain finance, DeFi, and beyond.

# “zkTLS” protocols

Verifier as a proxy in between the client and the server

Janus

 ZKPASS

 reclaim

 DECO  
Proxy mode

ORIGO

Verifier as an external entity



DIDO



 PADO

Crunch

DiStefano

 Shinkai

Requires server-side changes

TLS-N

draft-yasskin-http-origin-signed-responses-05

ROSEN

draft-cavage-http-signatures-11

Relies on trusted hardware

zkportal



Source: <https://bwetzel.medium.com/tls-oracles-liberating-private-web-data-with-cryptography-e66e5fad7c34>

# Open questions

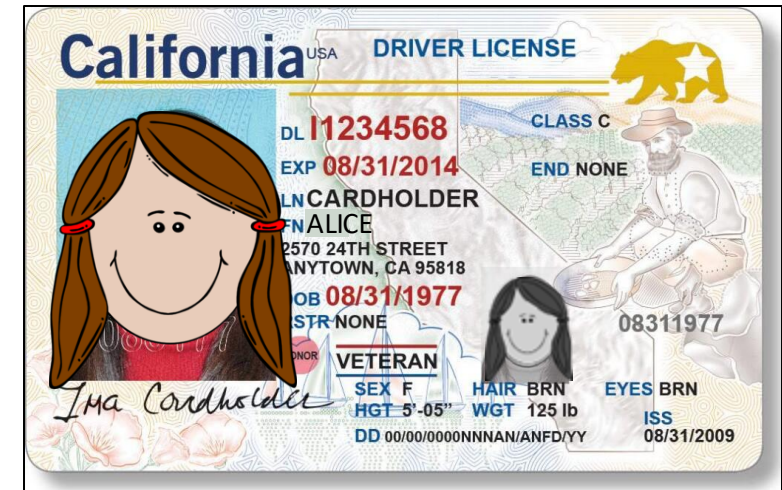
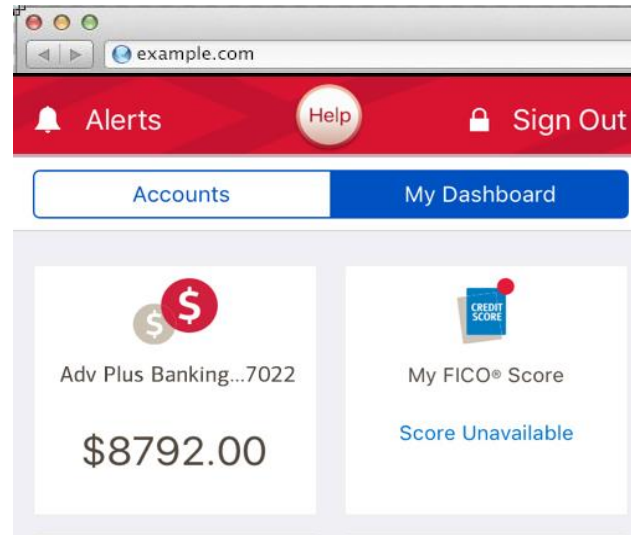
- Town Crier is publicly verifiable, but uses TEE
- DECO is purely crypto, but designated-verifier
- Directions to improve
  - Detecting & deterring TEE breaches (e.g., with incentives, 2PC between TEEs)
  - Making DECO-like protocols publicly verifiable (seems hard)

# Data Liberation

# Lots of important data locked up in web servers

API Key: b632dc74-  
8b87-45fd-8cfd-  
9a94cf216b46

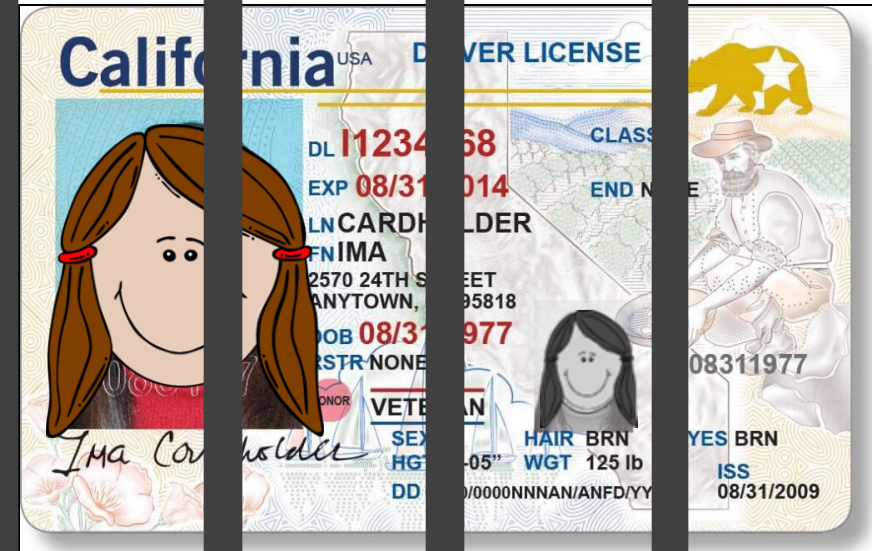
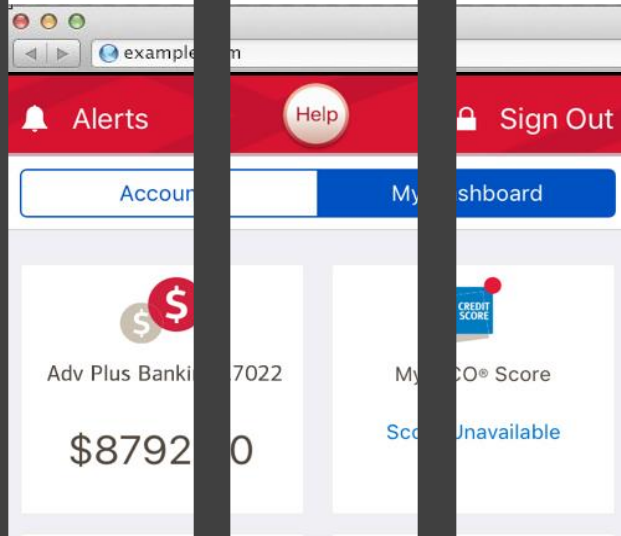
ETH-USD price: \$400



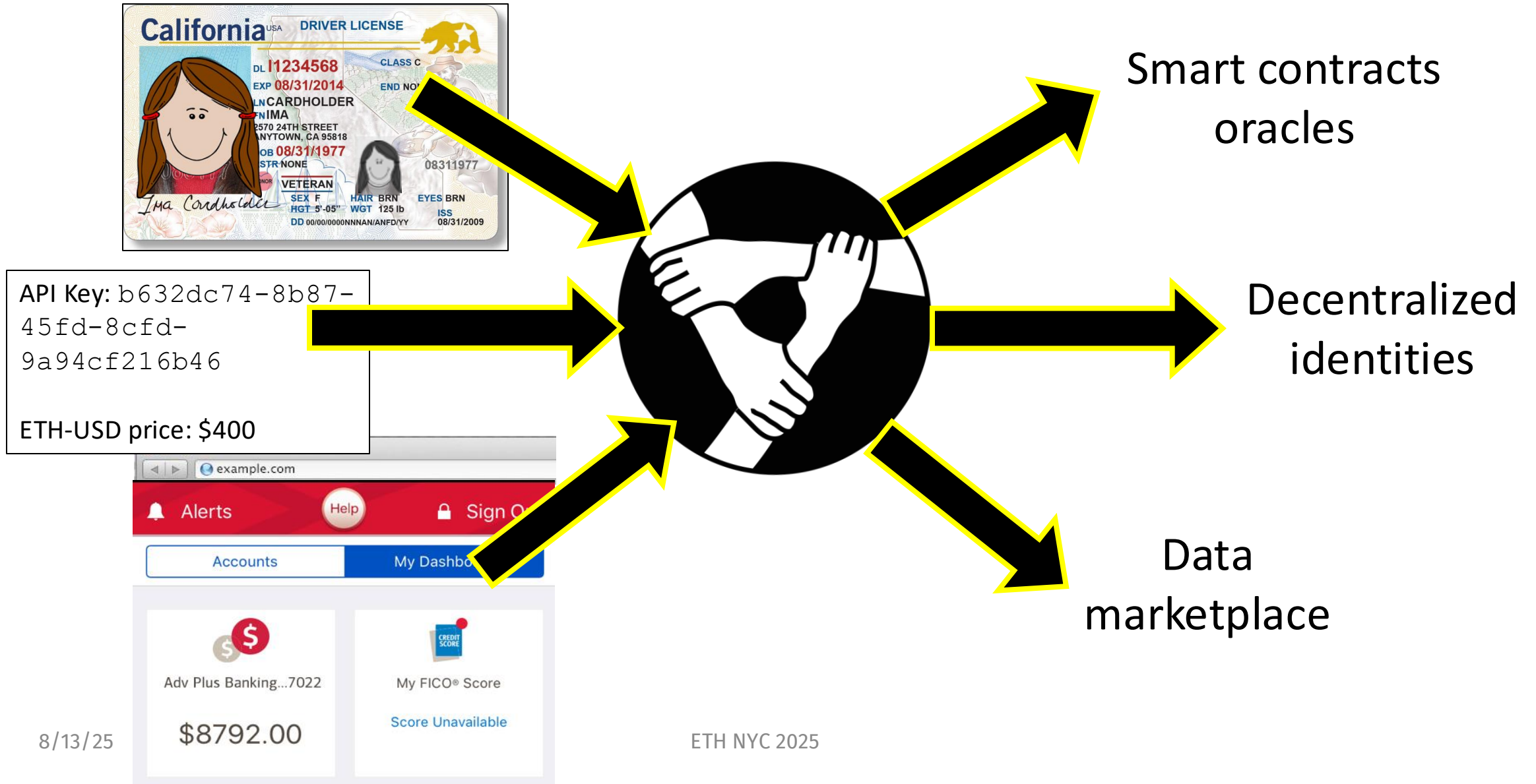
# Lots of important data locked up in web servers

API Key: b632dc74-  
8b87-451d-8c1d-  
9a94cf216b46

ETH-USD price: \$400



# Oracles liberates private web data





# How about some AI?

## Props for Machine-Learning Security

Ari Juels<sup>1</sup> and Farinaz Koushanfar<sup>2</sup>

<sup>1</sup>Cornell Tech

<sup>2</sup>University of California San Diego

October 29, 2024

### Abstract

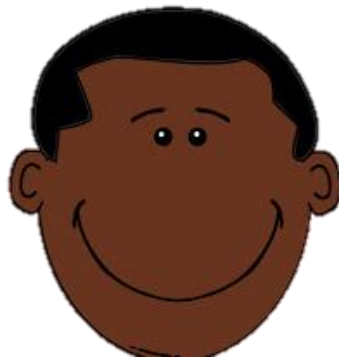
We propose *protected pipelines* or **props** for short, a new approach for authenticated, privacy-preserving access to deep-web data for machine learning (ML). By permitting secure use of vast sources of deep-web data, **props** address the systemic bottleneck of limited high-quality training data in ML development. **Props** also enable privacy-preserving and trustworthy forms of inference, allowing for safe use of sensitive data in ML applications. Finally, **props** offer a new approach to constraining adversarial inputs. **Props** are practically realizable today by leveraging privacy-preserving oracle systems initially developed for blockchain applications.

Many nice ideas on oracle's AI applications by Juels and Koushanfar (<https://arxiv.org/pdf/2410.20522>).



# How about *more* AI?

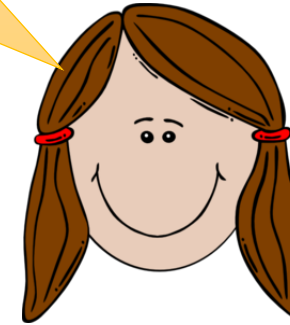
Verifier



Algorithm  
Audit

Here are my YouTube  
recommendations

Prover



Login: password

E.g., researchers collecting data

Researcher's goals:

- Verify data correctness
- Respect user privacy



# Summary: oracles

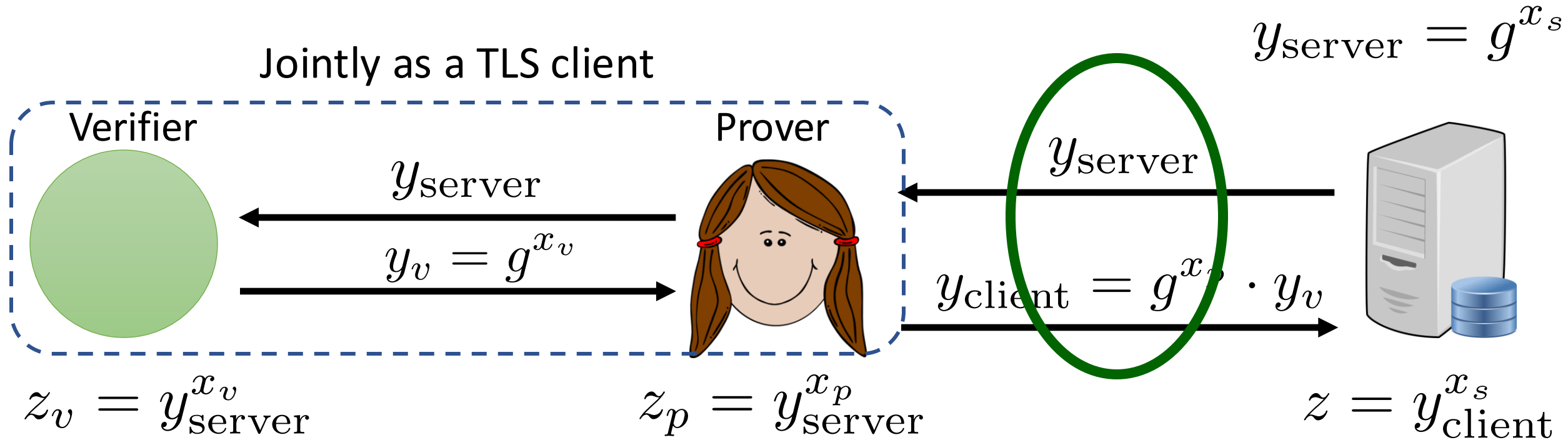


- Originated as systems to supply verifiable data to smart contracts, but their applications extend to **digital identity, social media, and AI**.
- [Town Crier](#) and [DECO](#) were among *the first* to formalize oracle security and realize it via verifiable provenance of TLS-encrypted data, turning HTTPS websites into sources of verifiable claims.
- These works initiated a new line of research and many real-world implementations.

- Website: <https://fanzhang.me>
- Twitter: 0xFanZhang
- Email: [f.zhang@yale.edu](mailto:f.zhang@yale.edu)

# Backup slides

# Diffie-Hellman (DH) -> Three-party DH

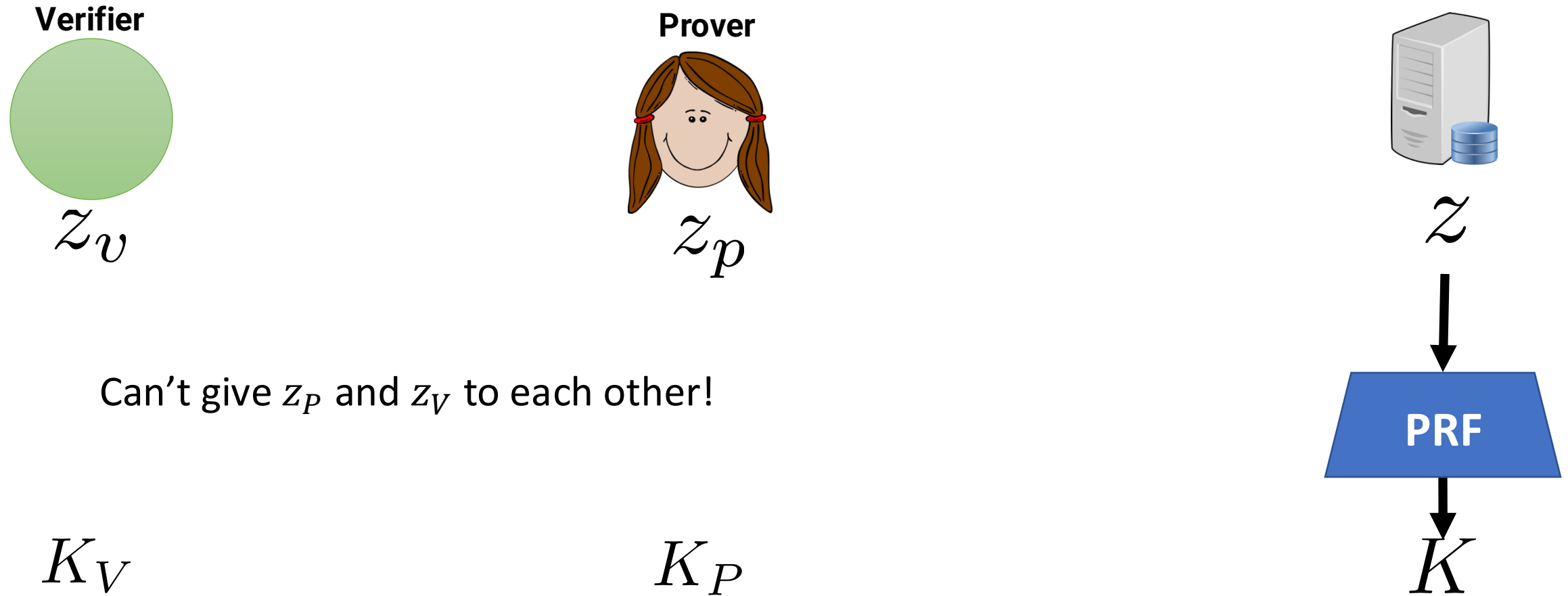


$$z_p \cdot z_v = z$$

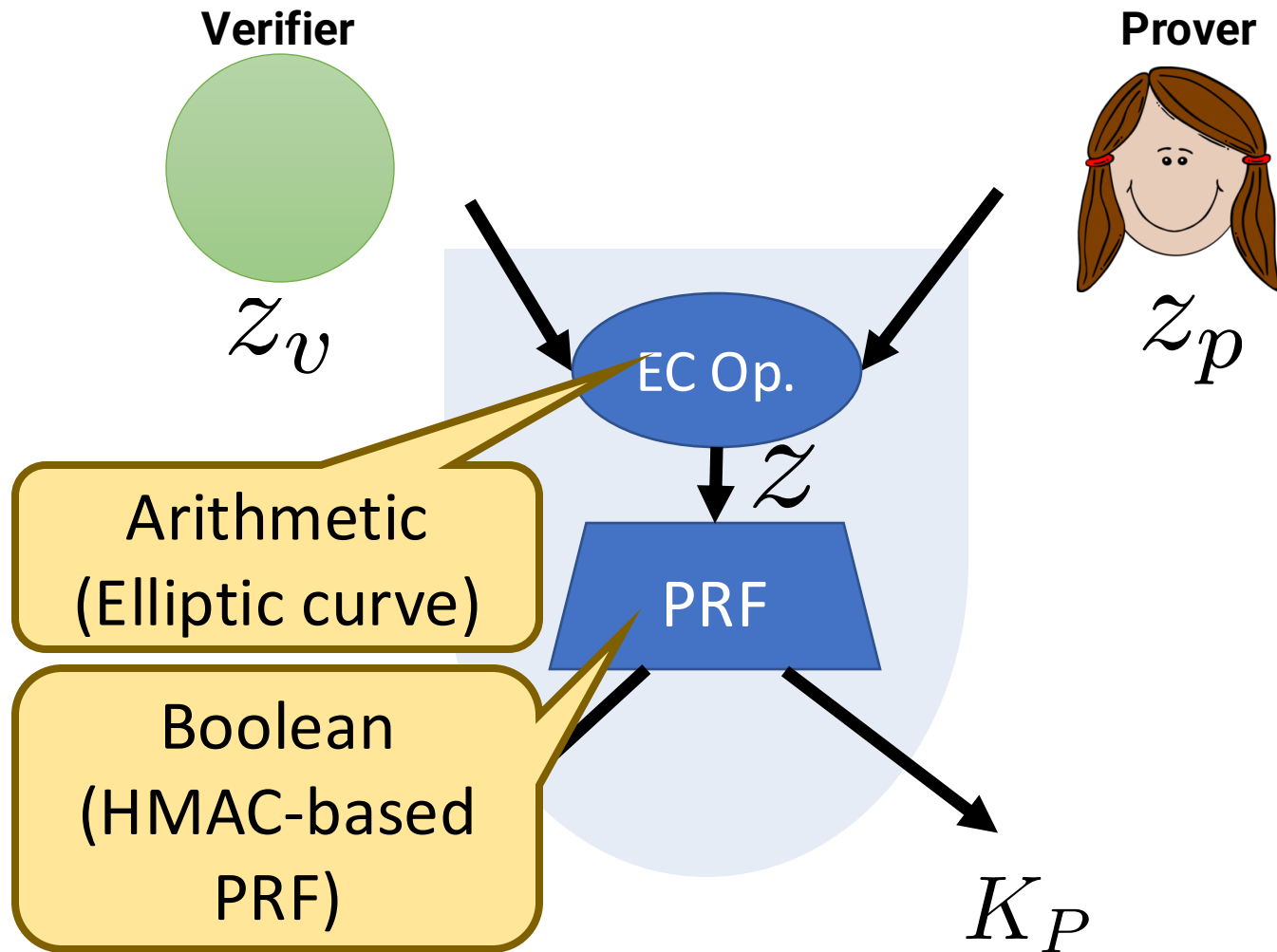
Elliptic curve  
(EC) points

( $z_p$  and  $z_v$  are “shares” of  $z$ )

# Three-party handshake: key derivation



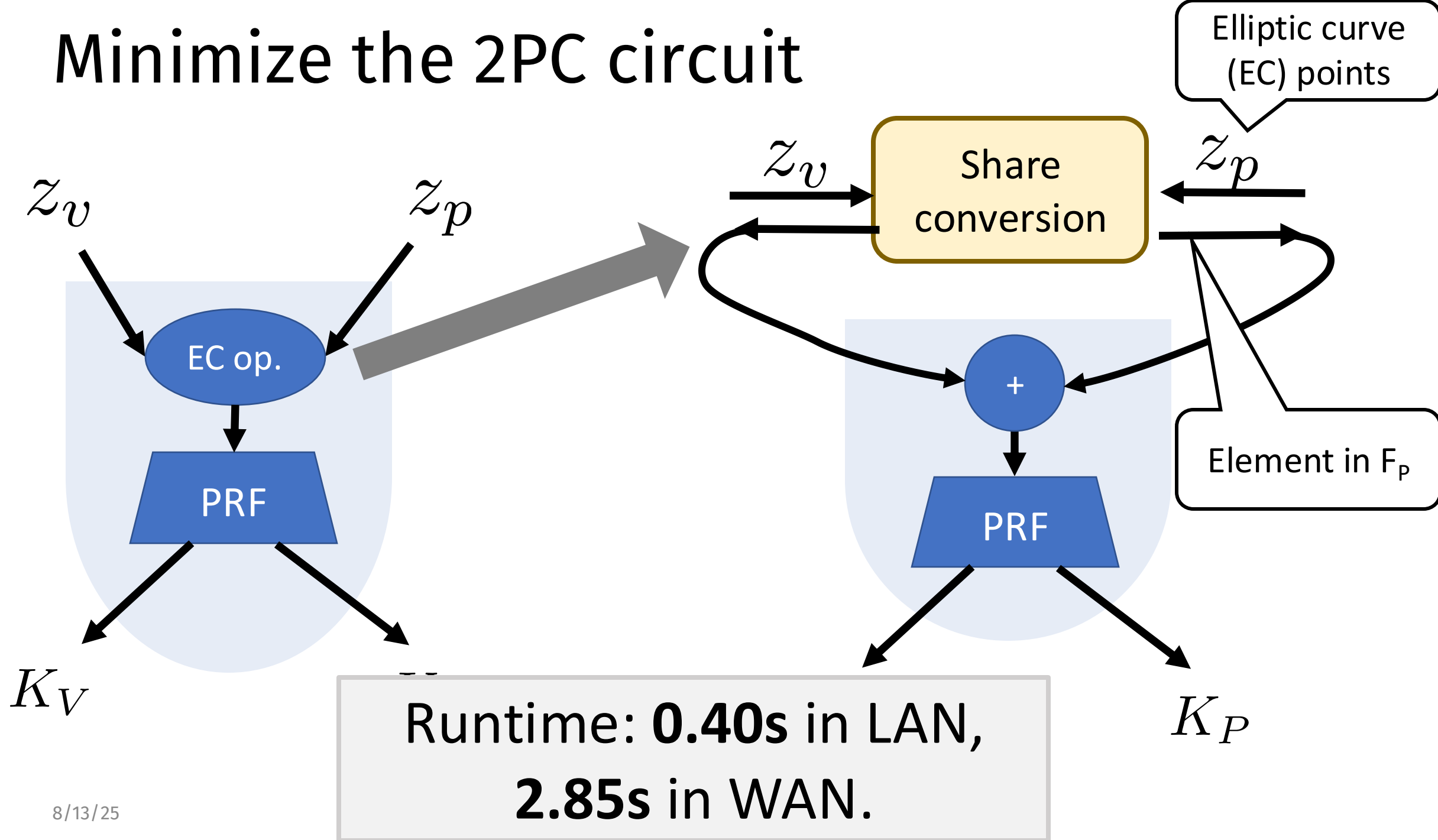
# Secure two-party computation (2PC)



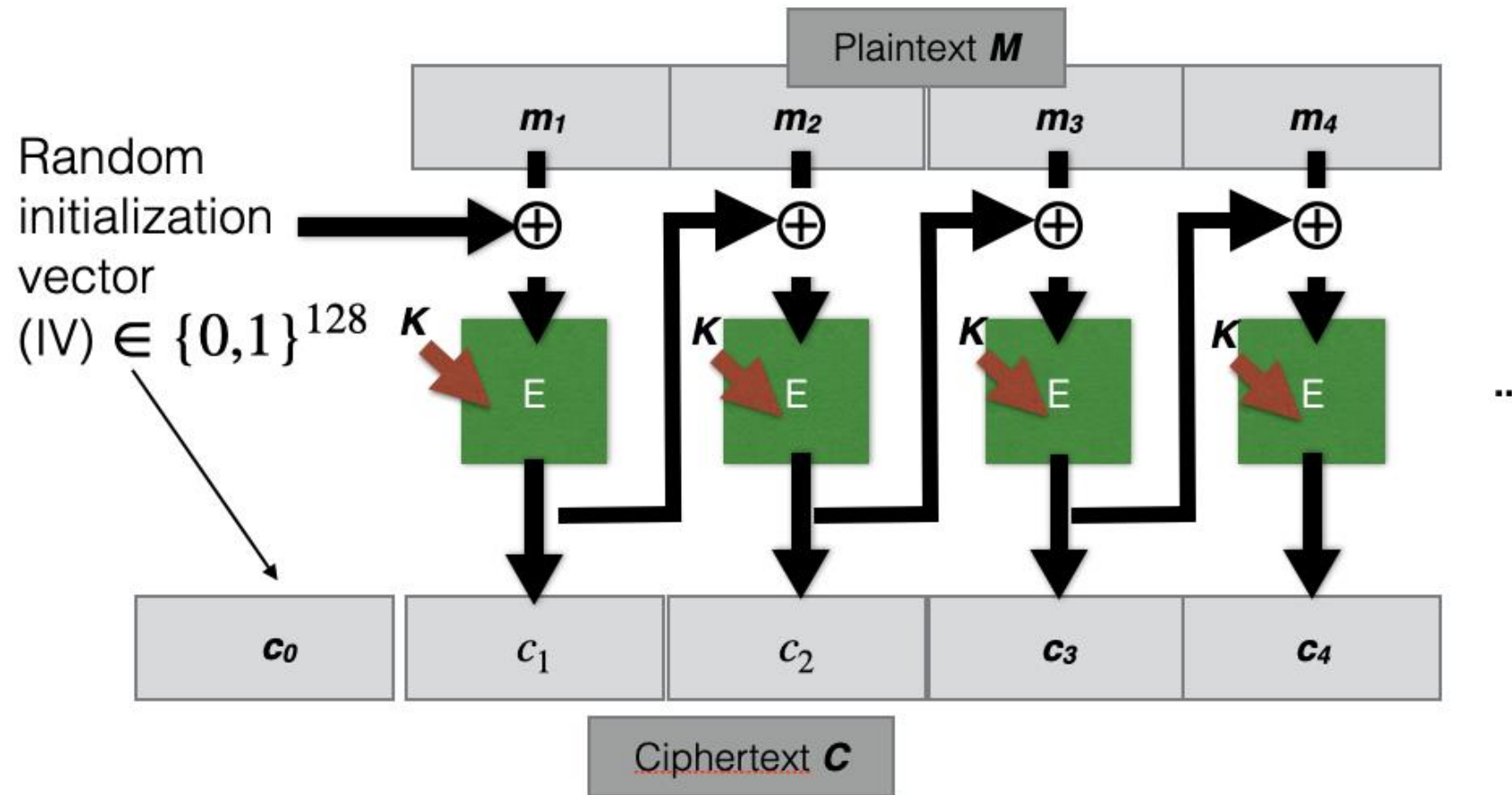
Two general approaches for

- **boolean** circuit (e.g., Yao82)
- Or **arithmetic** circuit (e.g., BGW88, CCD88)
- But **not both!**

# Minimize the 2PC circuit

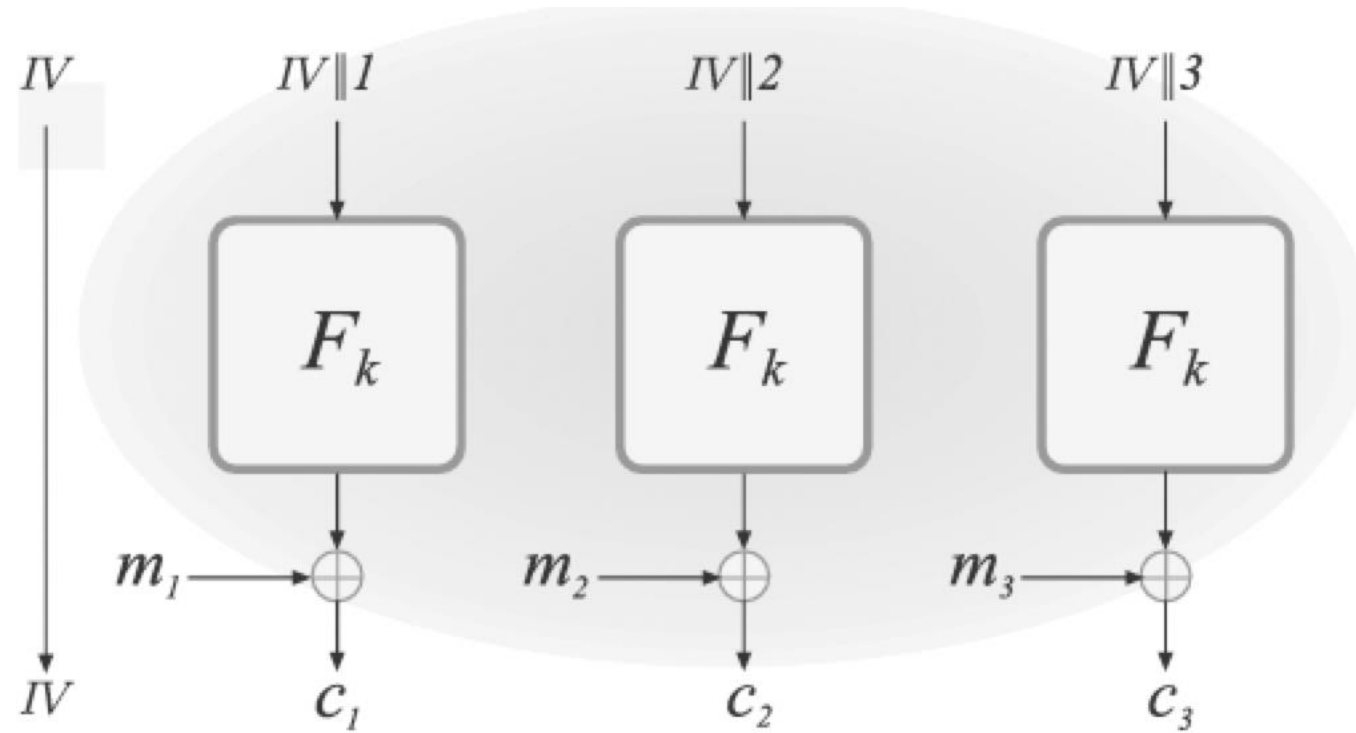


# CBC is not committing: Any ciphertext can be decrypted under any key.





# CTR mode is not committing (Any ciphertext can be decrypted under any key)



**FIGURE 3.9:** Counter (CTR) mode.

# What about CBC-HMAC?

Enc-then-MAC( $K_e, K_m, m$ ):

- $c = \text{Enc}(K_e, m)$
- $t = \text{Enc}(K_m, c)$
- Output  $(c, t)$

Attack:

- Output any  $\hat{k}_e \neq k_e$

Fix: derive  $K_e, K_m$  using a KDF, which TLS does, so we are good in case of CBC-HMAC.

# GCM mode is not committing

- GCM = AES-CTR + GMAC
- $\text{GMAC}(K, IV, c_1 || c_2 || \dots || c_m)$ :
  - $H = E(K, 0)$
  - $J = E(K, IV)$
  - Build a polynomial  $P(x) = c_1 x^{m-1} + c_2 x^{m-2} + \dots + c_m$  (in  $GF(2^{128})$ )
  - Return  $P(H) \oplus J$
- To break commitment is to find
  - $\text{GMAC}(K, IV, c_1 || c_2 || \dots || c_m) = \text{GMAC}(K', IV', c_1 || c_2 || \dots || c_m)$
- Attack: P is not collision resistant.