DECO: Liberating Web Data Using Decentralized Oracles for TLS

Cornell Tech & Cornell University
The 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS ’20)

Presentation by: Mina Petrovic, Bogdana Kolic & Iman Attia

Roadmap

1. Introduction

2. DECO protocol
   - Three-party handshake
   - Query execution
   - Proof generation

3. Applications
   - Confidential financial instruments
   - Anonymous credentials: Age proof
   - Price discrimination

4. Implementation and Evaluation

5. Conclusion

Roadmap

Why DECO?

I want to buy them. Let me check my bank account.

Alice (client-prover)

I want to sell you my homework solutions. But I need to know you have money to buy it.

Charlie (verifier)

Why DECO?

TLS

Bob (server -bank)

Alice (client-prover)

Charlie (verifier)

Why DECO?

TLS

Pinky promise

Do you **promise** you have the money?

Bob (server -bank)

Alice (client-prover)

Charlie (verifier)

Solutions before DECO

- Screenshots
  - Not anymore - banks have protections
  - Photo editing applications are powerful tool
- Sending Charlie her bank credentials - she would lose the money
- Forwarding TLS data to Charlie ⇔ screenshot
- Implement changes into TLS
- Use trusted hardware
  - Proven that it is not that trusted lately

PROBLEM: Leaking too much data, Alice only needs to prove that she has enough money, not show how much does she have

What is DECO?

- GOAL: Prove facts in zero knowledge with NO server modification
- HOW: Using three-party handshake → splitting the key between prover and verifier
  - Alice and Charlie generate SHARED key for TLS session with Bob
  - For Bob, the whole process is transparent
- OUTCOME:
  - Alice cannot forge the data from server
  - Alice can prove zero knowledge about data to Charlie

Roadmap

1. Introduction

2. DECO protocol
   - Three-party handshake
   - Query execution
   - Proof generation

3. Applications
   - Confidential financial instruments
   - Anonymous credentials: Age proof
   - Price discrimination

4. Implementation and Evaluation

5. Conclusion

Towards the DECO protocol

TLS 1.2 with CBC-HMAC

- Symmetric keys
- One key for encryption, one for the MAC tag

Strawman protocol

1. The prover sends all exchanged (encrypted) messages with the server to the verifier
2. Proves statements about server’s responses

Problem

- The prover holds session keys before sending messages to the verifier
    - It can forge arbitrary data (the commitment is not secure)

DECO protocol

Problem

- The prover holds session keys before sending messages to the verifier
  - It can forge arbitrary data (the commitment is not secure)

Key idea

- The prover and the verifier act as one client
- The prover learns the MAC key only after she commits

Shared MAC key algorithms

- Three-party handshake
- Query execution

Three-party handshake

Classic TLS

Elliptic curve Diffie-Hellman key exchange with ephemeral secrets (ECDHE) :

- Computation of a shared secret Z ∈ $EC(\mathbb{F}_p)$
- Evaluation of the TLS-PRF function with Z to derive the session key k

1 **Key exchange**

P and *V* compute their share of the secret Z:

$Z = Z_P + Z_V$

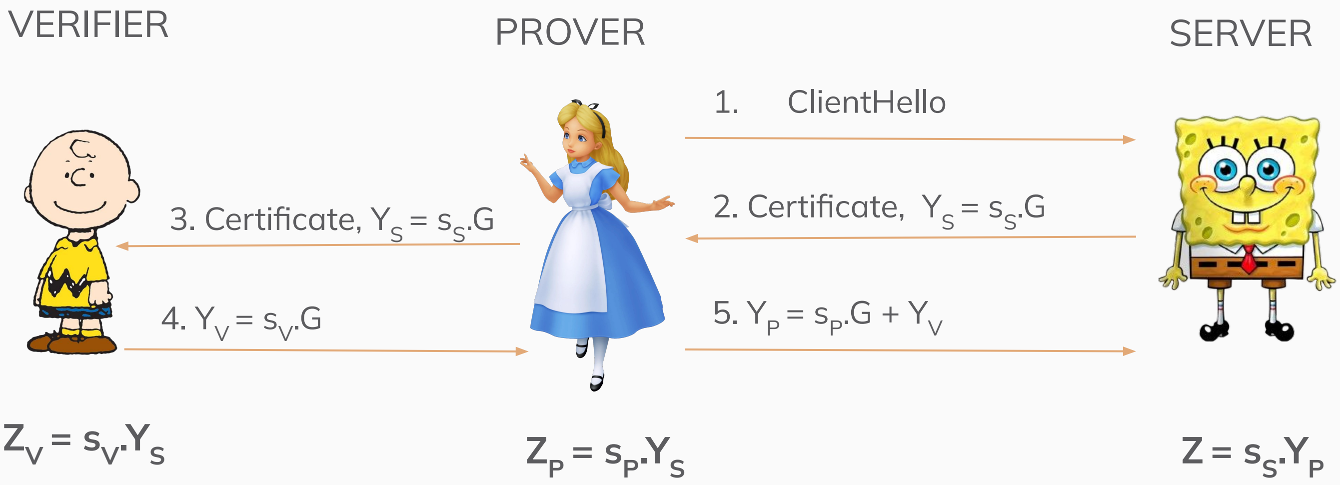
2 **Key derivation**

P an V evaluate the TLS-PRF function on Z_P and Z_V to derive their share of the session key:

$k = k_P + k_V$

Three-party handshake - key exchange

Reminder: we are working with EC($\mathbb{F}_P$), the generator is G



Three-party handshake - key derivation

P and *V* evaluate the TLS-PRF on their share of Z

... but, addition of EC($\mathbb{F}_p$) points in a boolean circuit is costly - approximately, we need over 900,000 AND gates

Optimization - compute additive shares of one coordinate

Now, addition of two $\mathbb{F}_p$ points has AND complexity of about 3|p|

ECtF function

For computation of the additive shares of the x-coordinate of Z.
Takes Z_P and Z_V as inputs, each party learns its additive share (in $\mathbb{F}_p$)

Query execution

1. **After the 3P-HS, the prover holds the encryption key k^{ENC}, and the prover and the receiver hold their secret shares k_P^{MAC} and k_V^{MAC} of the MAC key k^{MAC}**

2. **The prover sends the query Q to the server as a standard TLS client**

   A 2PC protocol between *P* and *V* is used to compute the MAC tag τ without revealing k^{MAC} to *P*, then *P* uses k^{ENC} to encrypt (Q||τ) and sends it to the server

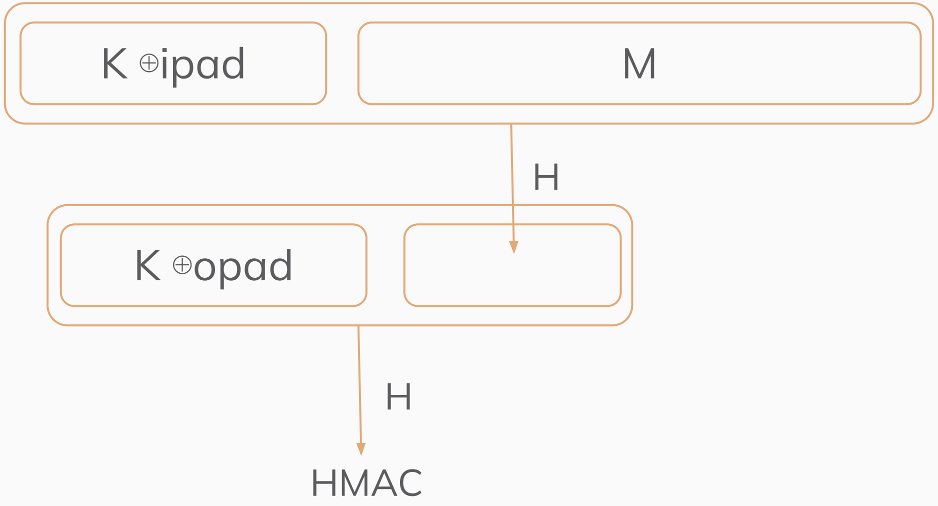
3. **2PC is expensive for large queries**

   We need to optimize the MAC tag computations

Query execution

Recall the formula for computing HMAC of message M with key K:

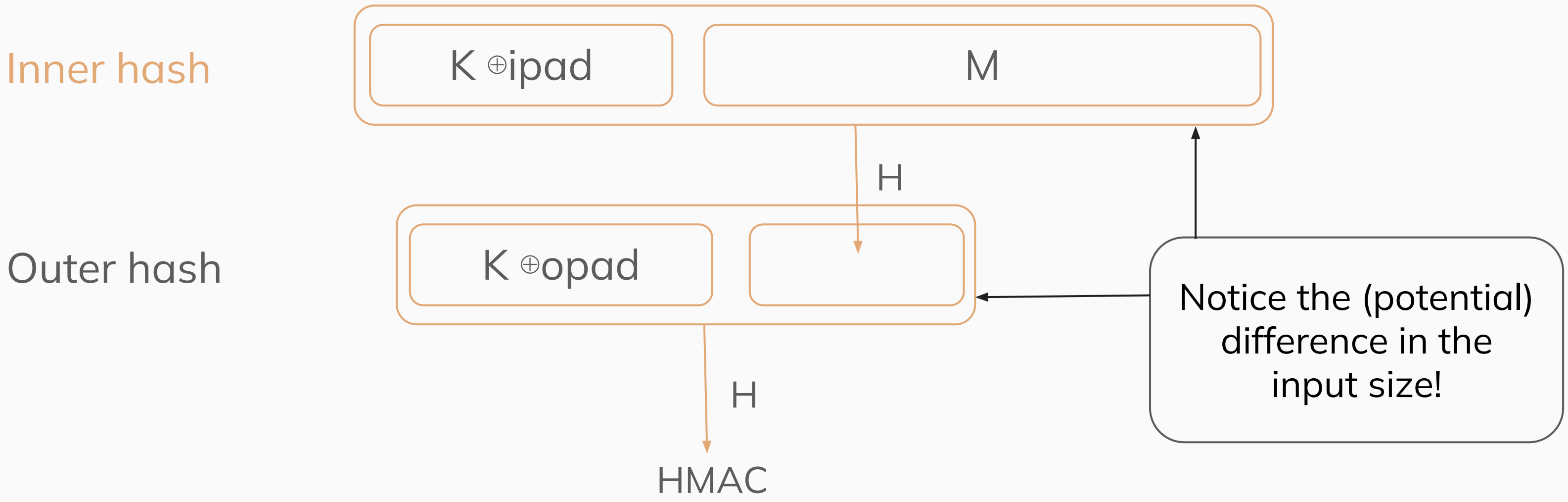
$HMAC_H(K, M) = H((K \oplus opad) \| H((K \oplus ipad) \| M))$ (H stands for SHA-256)



Query execution

Recall the formula for computing HMAC of message M with key K:

$HMAC_H(K, M) = H((K \oplus opad) \| H((K \oplus ipad) \| M))$ (H stands for SHA-256)

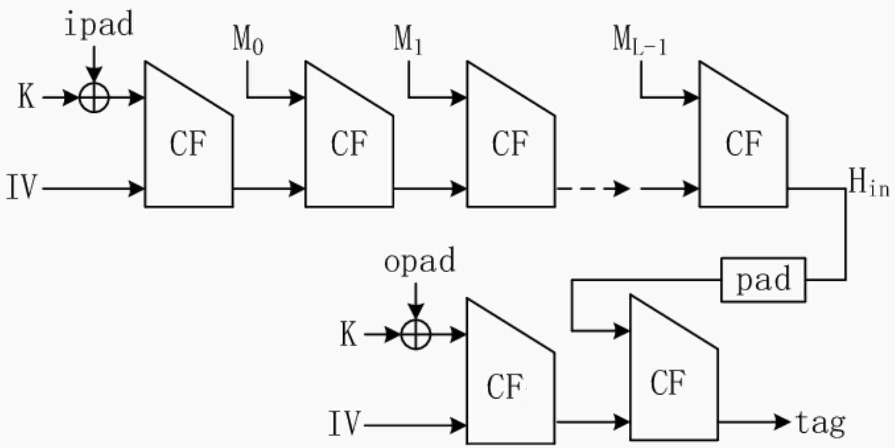


Query execution

Recall the formula for computing HMAC of message M with key K:

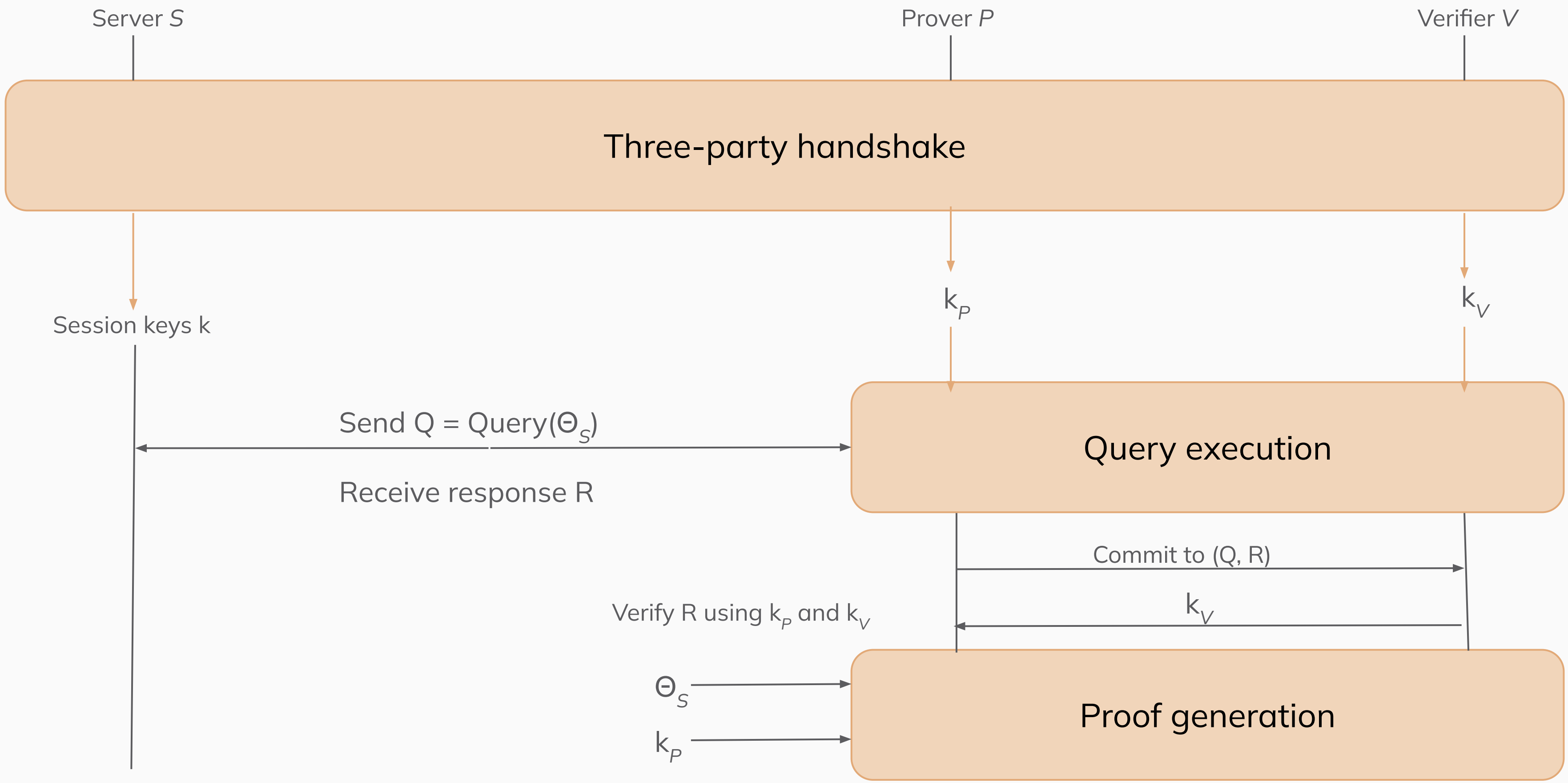
$HMAC_H$(K, M) =H((K⊕opad) || H((K⊕ipad) || M))

(H stands for SHA-256, CF is the one-way compression function)



Recall: The motivation for using 2PC is hiding the secret key K from the prover

https://www.researchgate.net/figure/The-HMAC-SHA-256-construction_fig1_307694142


Server S
Prover P
Verifier V
Three-party handshake
Session keys k
k_P
k_V
Send Q = Query(Θ_S)
Receive response R
Query execution
Commit to (Q, R)
k_V
Verify R using k_P and k_V
Θ_S
k_P
Proof generation

Proof generation

1 **Selective opening**

- ***Reveal mode*** - reveal **only a certain chunk** of the plaintext to the verifier
- ***Redact mode*** - reveal the plaintext **without a certain chunk** to the verifier

2 **Context integrity by two-stage parsing**

- ***Context integrity*** - prove that the revealed substring is produced in a certain way expected by the verifier
- ***Two-stage parsing*** - Reduce the cost of proving context integrity by preprocessing the server's response (*P* and *V* both agree on the transformation that is to be used)

Roadmap

1. Introduction

2. DECO protocol
    - Three-party handshake
    - Query execution
    - Proof generation

3. Applications
    - Confidential financial instruments
    - Anonymous credentials: Age proof
    - Price discrimination

4. Implementation and Evaluation

5. Conclusion

1. Confidential financial instruments

- Financial derivatives are commonly cited in smart contract applications, emphasizing the need for authenticated data feeds (e.g., stock prices).

- Reminder: **A smart contract** is a self-executing digital contract where the terms of the agreement are written directly into code.

- One popular financial instrument that is easy to implement in a smart contract is **a binary option.**

1. Confidential financial instruments

Charlie

Alice

Binary Option?

A contract between two parties betting on whether, at a designated future time (e.g., the close of day **D**), the price P^* of some asset N will equal or exceed a predetermined target price P. The contract condition is: $P^* \geq P$

1. Confidential financial instruments

Past Approach: Mixicle

Mechanism:
The oracle O can conceal the underlying asset ***N*** and target price ***P*** for a binary option on chain. It simply accepts the option details off chain, and reports only a bit specifying the outcome ***Stmt := P* ≥? P***

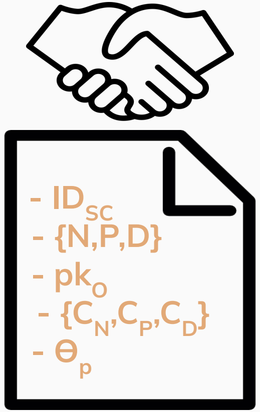
Limitation:
A limitation of a basic Mixicle construction is that the oracle $\mathcal{O}$ itself learns the details of the financial instrument. Prior to DECO, only oracle services that use TEE could conceal queries from the oracle.

1. Confidential financial instruments

1. Setup:

Charlie

Alice

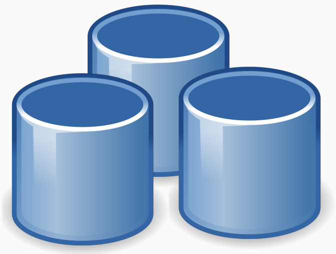


- ID_{SC}: Smart Contract ID
- $\{sk_O, pk_O\}$: denote the oracles' key pair
- {N,P,D}: binary option
- $\{C_N, C_P, C_D\}$: commitments
- Θ_p the URL to retrieve asset prices

1. Confidential financial instruments

2. Settlement:

VERIFIER
Oracle O



ZKP using DECO

$S = Sig(sk_O, ID_{SC})$

PROVER
Alice (Winner)

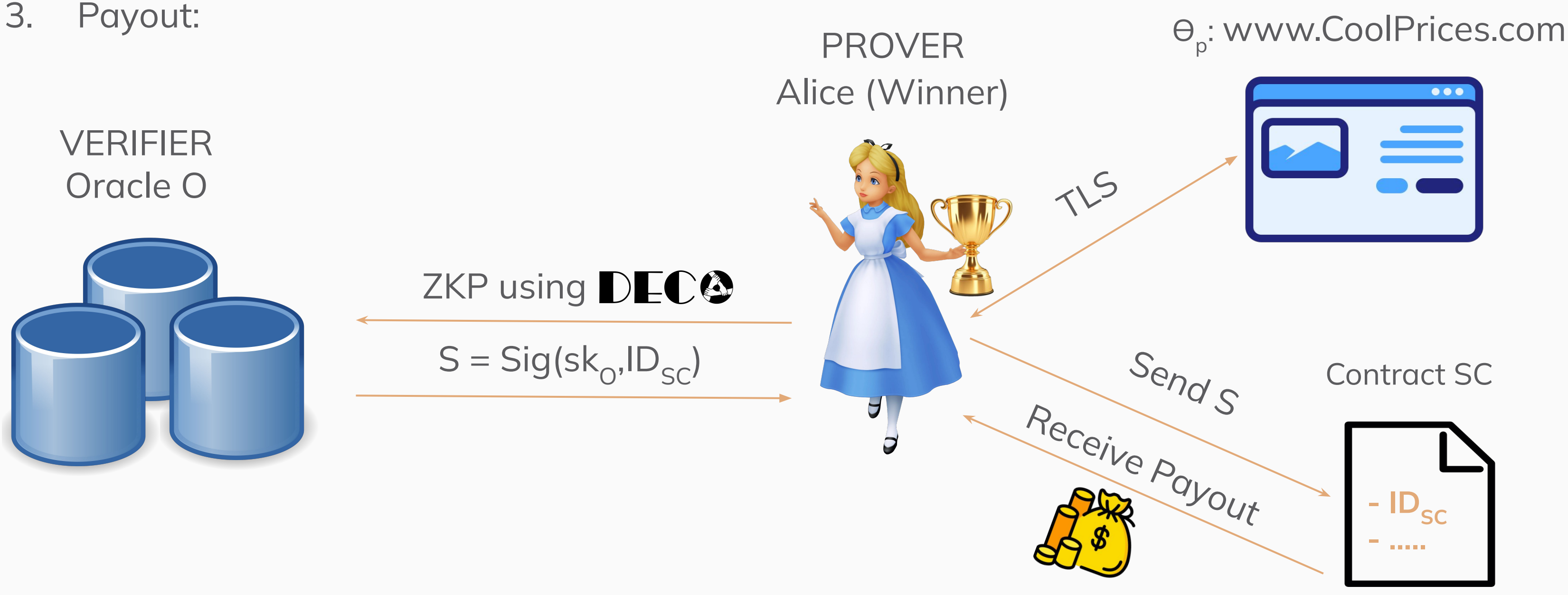
TLS

θ_p: www.CoolPrices.com

```
> GET
/query?function=GLOBAL_QUOTE&
 ↩→ symbol= GOOGL
Host: www.CoolPrices.com
 >
 { "Global Quote": { "01. symbol":
"GOOGL", "05. price": "1157.7500",
"07. day": "2019-07-16"}}
```

1. Confidential financial instruments

3. Payout:



1. Confidential financial instruments

Implementation Details - Two Stage Parsing Scheme:

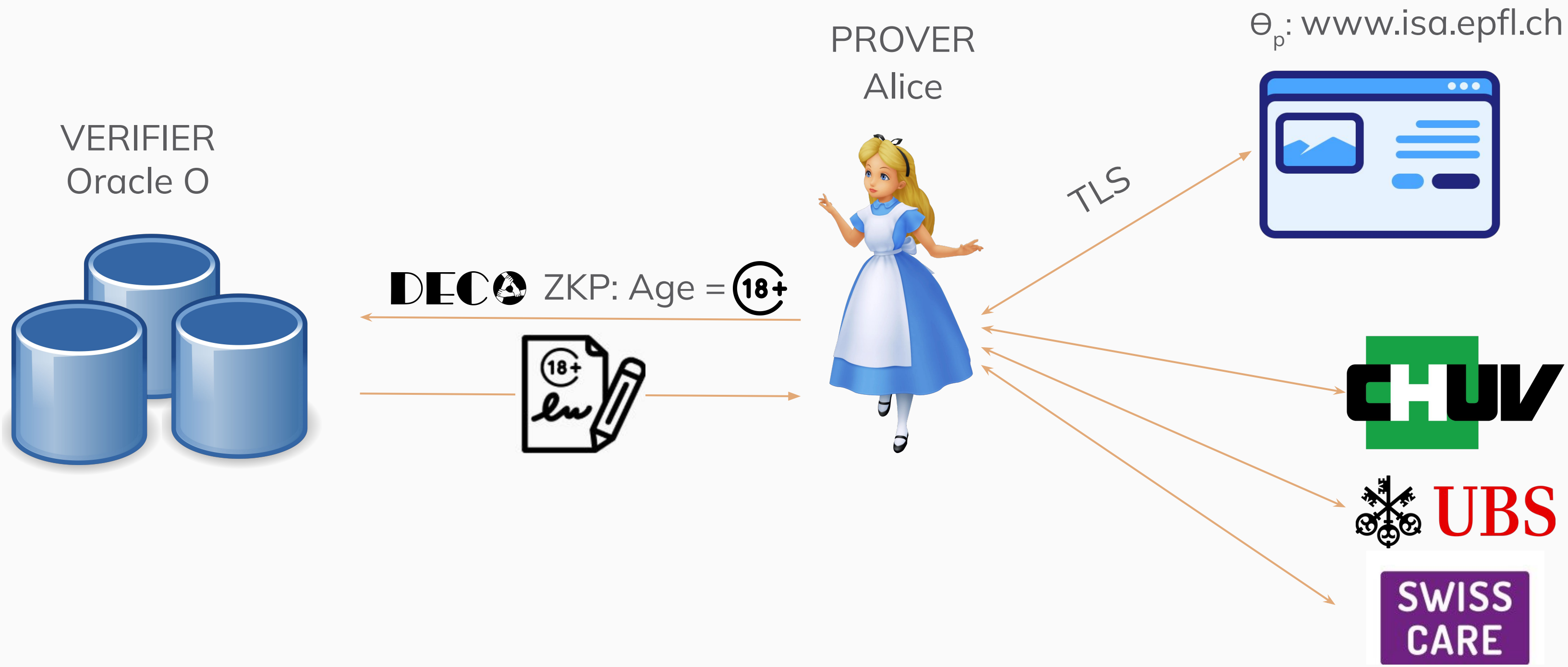
First Stage:
Party P parses the response R locally and identifies the smallest substring that can convince Party V.

Second Stage:
Party P proves knowledge of (R_{price}, P, r_P) in Zero-Knowledge (ZK), ensuring the following:

1. R_{price} is a substring of the decrypted ciphertext Ŕ.
2. The price starts with “05. price”.
3. The subsequent characters form a floating-point number P^*, and that $P^* \geq P$.
4. $com(P, r_P) = C_P$, where C_P is the commitment for price P.

Using the CBC-HMAC cipher suite, the ZKP circuit involves redacting the entire record, computing commitments, and performing string processing.
Secure? Unique keys!

2. Anonymous credentials: Age proof



3. Price discrimination

Price Discrimination: Same product sold at different prices to different buyers based on tracking data (e.g., zip codes).

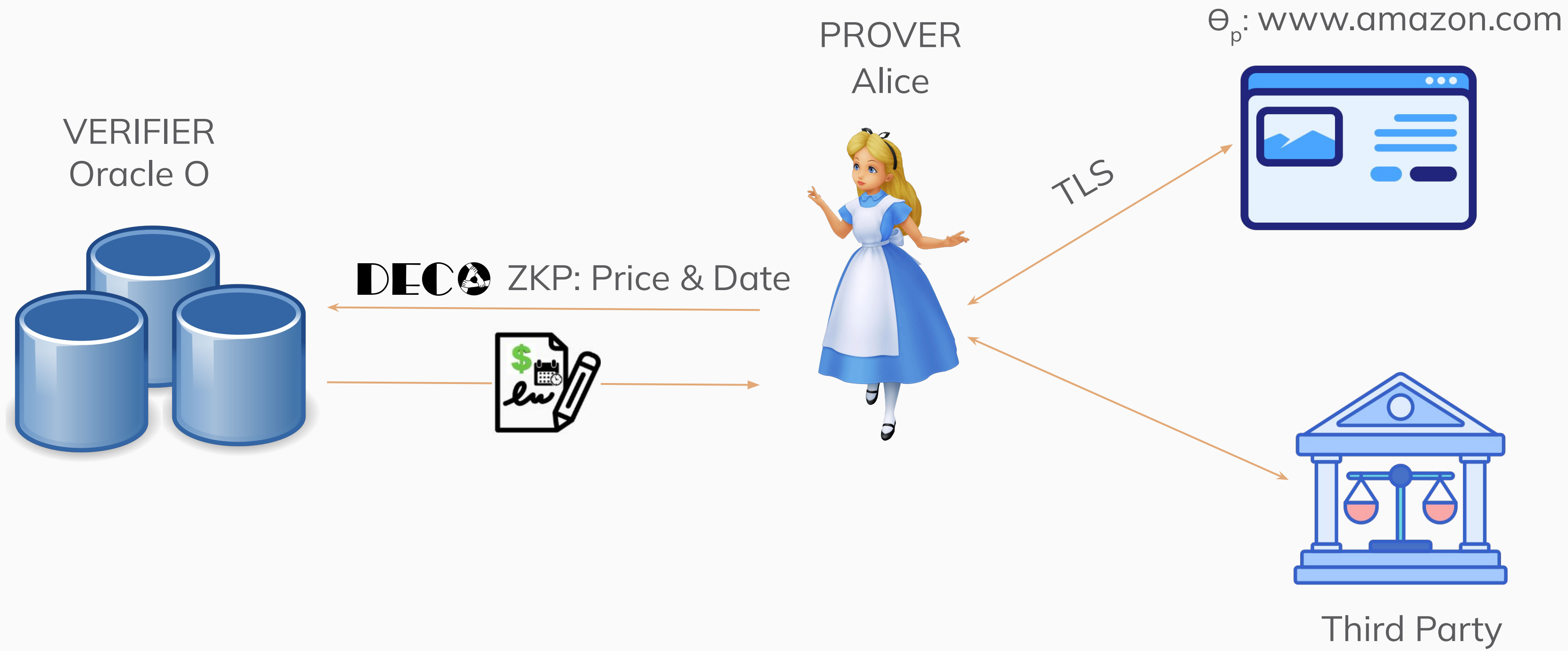
Legal Aspects: Permissible unless it harms competition (U.S. CFT laws); impacted by privacy laws (e.g., GDPR in Europe).

DECO's Solution: Allows buyers to verify price discrimination claims while keeping personal info hidden.

AES-GCM Cipher: Uses AES-GCM cipher suite and Reveal mode to reveal only necessary order details.

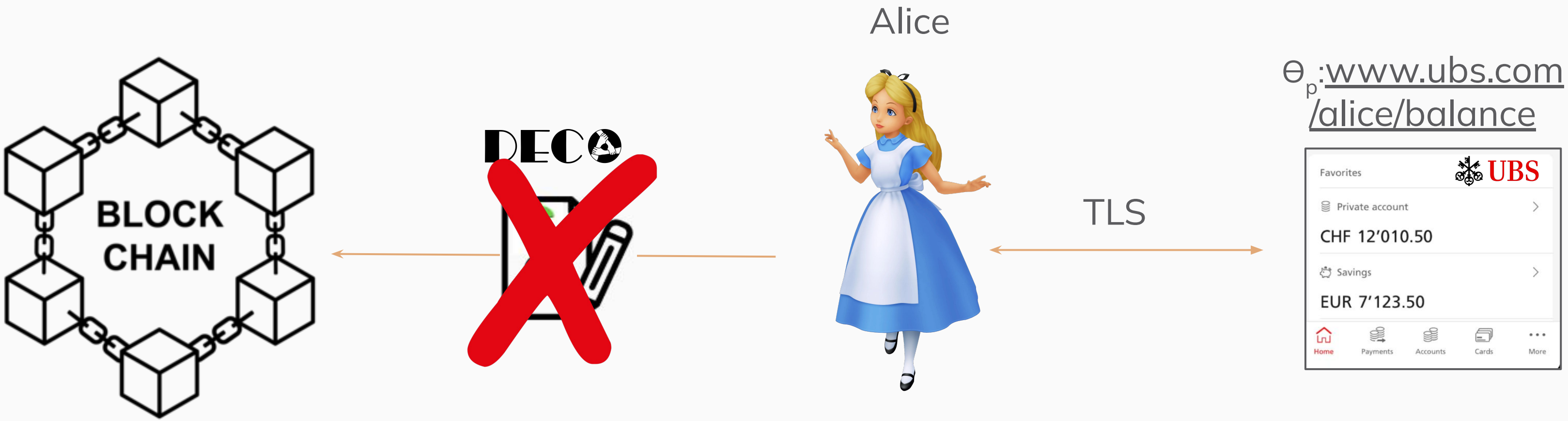
```
<table >
        <tr>Order Placed: November 23,
2018</tr>
        <tr>Order Total: $34.28</tr>
        <tr>Items Ordered: Food
Processor</tr>
</table >
...
<b> Shipping Address: </b>
<ul class="displayAddressUL ">
        <li class="FullName">Alice</li>
        <li class="Address">Wonderland</li>
        <li class="City">New York</li>
</ul>
```

3. Price discrimination



Limitations of DECO in practice

- DECO can't generate ZK proofs directly without having the Oracle Network.
- Alice and Charlie need to trust O for integrity.



Roadmap

1. Introduction

2. DECO protocol
   - Three-party handshake
   - Query execution
   - Proof generation

3. Applications
   - Confidential financial instruments
   - Anonymous credentials: Age proof
   - Price discrimination

4. Implementation and Evaluation

5. Conclusion

Implementation

- Implemented in ~4700 lines of C++ code
- Three-party handshake (3P-HS) protocol for TLS 1.2.
- Two-party computation protocols (2PC-HMAC, 2PC-GCM).
- Uses Relic (Paillier cryptosystem) and EMP toolkit for secure computation.
- Integrated with mbedTLS for end-to-end TLS session security.
- Zero-knowledge proofs were implemented using libsnark, with statement templates adapted for specific applications via SNARK compilers like xjsnark.

Evaluation

Table 1: Run time (in ms) of 3P-HS and query execution protocols.

		LAN		WAN	
		Online	Offline	Online	Offline
3P-Handshake	TLS 1.2 only	368.5 (0.6)	1668 (4)	2850 (20)	10290 (10)
2PC-HMAC	TLS 1.2 only	133.8 (0.5)	164.9 (0.4)	2520 (20)	3191 (8)
2PC-GCM (256B)	1.2 and 1.3	36.65 (0.02)	392 (8)	1208.5 (0.2)	12010 (70)
2PC-GCM (512B)	1.2 and 1.3	53.0 (0.5)	610 (10)	2345 (1)	12520 (70)
2PC-GCM (1KB)	1.2 and 1.3	101.9 (0.5)	830 (20)	4567 (4)	14300 (200)
2PC-GCM (2KB)	1.2 and 1.3	204.7 (0.9)	1480 (30)	9093.5 (0.9)	18500 (200)

Evaluation

Table 2: Costs of generating and verifying ZKPs in proof-generation phase of DECO for applications in Sec. 6.

	Binary Option	Age Proof	Price Discrimination
prover time	12.97 ± 0.04s	3.67 ± 0.02s	12.68 ± 0.02s
verifier time	0.01s	0.01s	0.05s
proof size	861B	574B	1722B
# constraints	617k	164k	535k
memory	1.78GB	0.69GB	0.92GB

Roadmap

1. Introduction

2. DECO protocol
   - Three-party handshake
   - Query execution
   - Proof generation

3. Applications
   - Confidential financial instruments
   - Anonymous credentials: Age proof
   - Price discrimination

4. Implementation and Evaluation

5. Conclusion

Conclusion

I have no idea what are they doing!

I only have eyes for Alice

Bob (server -bank)

TLS

Let's share a key!

Alice (client-prover)

DECO

No need for pinky promise, we have DECO!!!!

Charlie (verifier)

Thanks!

Do you have any questions?