The Phish-Market Protocol

Securely Sharing Attack Data Between Competitors

Tal Moran       Tyler Moore

Center for Research on Computation and Society
Outline

• Motivation
• Challenges
• The Phish-Market Protocol
  – Concepts, not math
• A little math [Optional]
• Implementation
Motivation

• Phishing is a serious problem for banks
• Phishers set up fake websites:
  – pretend to be banks
  – scam users into entering passwords
  – links often appear in spam
Motivation

• Banks hire `take-down’ companies to patrol internet for phishing sites
  – Aggregate multiple URL feeds
  – Read from public sources (e.g., APWG)
  – Proprietary sources (e.g., spam honey traps)
    • Considered competitive advantage
• Take-down companies compete for clients
• Moore and Clayton estimate $330,000,000 cost of refusing to share data
  – For these two companies alone!
The Proposal

• Create a market for phishing data
  – Remunerate companies for sharing data
  – Must take into account competition
Challenges

- Buyer learns only URLs that phish client banks
- Seller does not learn about Buyer’s clients
- Buyer pays for new each URL learned
- Buyer doesn’t pay for URLs already known

Solved!

-In Theory: “Secure Computation”

In Practice:
- Generic solutions extremely inefficient

- Can’t introduce significant delays
Protocol Ideas

• Idea: “pay” with encrypted “coins”

• Reveal only payment totals
  – Can’t tell which URLs were those “sold”.

• relaxations for efficiency:
  – Buyer learns “tags” of all Seller URLs
  – Buyer learns already known URLs
    (but does not pay)
Transaction Overview

1. Seller offers URL to Buyer
   – Oblivious Transfer

2. Buyer sends encrypted payment
   – Homomorphic Commitment

3. Buyer “proves” payment is good
   – Zero-Knowledge Proof

4. Buyer “proves” he knew URL
   – Zero-Knowledge Proof

• Seller’s view is always the same!
The Phish-Market Protocol

• Meet Sally and Bob:
Commitment Schemes

- Commitment to a value:
  - Commit now
    - "Hiding": Sally doesn’t learn contents
  - Reveal later
    - "Binding": Bob can’t change the contents
    - Bob commits in advance to the URLs he knows
Zero-Knowledge Equivalence Proofs

- Prove two commitments are the same
- Don’t reveal anything else

- To prove payment is good: “payment=C(1)”
- To prove Bob already knew URL
Zero-Knowledge Equivalence Proofs with trapdoor

- Sometimes Bob shouldn’t pay
- Sometimes Bob didn’t know URL beforehand

- Trapdoor lets Bob use secret key to fake proof
- Sally can’t tell the difference
Oblivious Transfer (OT)

- Sally prepares two encrypted items
- Bob gets to choose only one encryption key
  - Either learn URL or get extra “proof key”
- Sally doesn’t learn which
  - assume keys are indistinguishable
Homomorphic Addition

• Special commitment scheme:
  – Can add commitments without opening them

This is a payment commitment

(A chest won’t fit in the piggy bank)
Homomorphic Addition

• Special commitment scheme:
  – Can add commitments without opening
  – Can reveal sum without revealing anything else
High-Level Protocol Summary

Commit to previously known URLs
URL Tag, C(URL) and single ZK proof key

URL
2^{nd} proof key

OT

Choice result

\( e = \text{Commitment to payment} \)
\( u = \text{Commitment to URL} \)

ZK Proof 1:
\( e = C(1) \)

ZK Proof 2:
\( u = C(\text{URL}) \)

Proof 3: \( u \) is in committed set
High-Level Protocol Summary

Commit to previously known URLs
URL Tag, C(URL) and single ZK proof key

URL
2\textsuperscript{nd} proof key
\texttt{OT}

\begin{align*}
e &= \text{C}(0) \\
u &= \text{C}(\text{Fake previously known URL})
\end{align*}

\begin{align*}
\text{ZK Proof 1} \\
e &= \text{C}(1) \\
u &= \text{C}(\text{URL})
\end{align*}

Proof 3: \( u \) is in committed set
High-Level Protocol Summary

Commit to previously known URLs
URL Tag, C(URL) and single ZK proof key

Want URL
URL

OT

URL
2nd proof key

Not in list

\[ e = C(1) \]
\[ u = C(\text{Fake previously known URL}) \]

ZK Proof 1
\[ e = C(1) \]

ZK Proof 2
\[ u = C(\text{URL}) \]

Proof 3: u is in committed set
High-Level Protocol Summary

Commit to previously known URLs
URL Tag, C(URL) and single ZK proof key

URL
2nd proof key

OT

URL

Want URL

Is in List

\[ e = C(0) \]
\[ u = C(URL) \]

ZK Proof 1
\[ e = C(1) \]

ZK Proof 2
\[ u = C(URL) \]

Proof 3: u is in committed set
Formal Security Guarantees

• For Seller:
  – Equivalent to an “ideal world” with a trusted third party.

• For Buyer:
  – Seller doesn’t learn anything about Buyer’s secrets except what is revealed by aggregate payment.

• Theorem: the protocol is secure!
Proof 3: Merkle Trees

• Efficient commitment to large sets
  – Send only the root of the tree:

• Proofs are not zero-knowledge
  – We use commitments as leaves
  – Add “chaff” commitments
ZK Equivalence Proof
(for homomorphich commitments)

• To prove: \( C(x) \approx C(y) \)
  – Reduce to “proof of committed value”:
  – Prove: \( \frac{C(x)}{C(y)} = C(x-y) \approx C(0) \)

• Standard protocol to prove \( C(x) \approx C(0) \):
  1. Prover commits: \( C(b) \), sends \( b \)
  2. Verifier sends random challenge: \( a \)
  3. Prover opens commitment: \( C(ax+b) = C(x)^a C(b) \)
     • Value must be: \( b \)

• If \( x \neq 0 \), w.h.p. (over \( a \)) we have: \( ax + b \neq b \)
• If Prover knows \( a \), can cheat by computing \( b' = ax + b \) in step 1.

Note:
• If \( x \neq 0 \), w.h.p. (over \( a \)) we have: \( ax + b \neq b \)
• If Prover knows \( a \), can cheat by computing \( b' = ax + b \) in step 1.

 Doesn’t open commitment

Note:
• Arithmetic is modular!


Trapdoor ZK Proofs

• **ZK $\sum$- Protocol:**
  1. Prover commits
  2. Verifier sends a random challenge
  3. Prover opens commitment

• **Generic transformation to add trapdoor:**
  1. Prover commits
  2. Challenge computed using Coin-Flipping protocol
  3. Prover opens commitment

• We use Coin-Flipping protocol with trapdoor.
Blum Coin-Flipping
(with trapdoor)

- Use a commitment to flip a coin:
  - Bob chooses a random value
    - He’s committed, but Sally doesn’t know the value
  - Sally chooses a random value
  - Bob opens his commitment.
  - The value of the coin is the sum.
- Bob can cheat if he can equivocate on commitment
Our Implementation

• Pedersen Commitment
• Naor-Pinkas Oblivious Transfer
  – (uses “Random Oracle”)
• Both based on hardness of discrete log in a generic group
  • can be implemented over Elliptic-Curves or using modular arithmetic
Performance

• Elliptic-Curve based Java implementation
• Ran experiments using real data (two weeks)
• ~10000 URLs
• Avg. 5 sec delay.
• Max. 35 sec.
The Qilin Crypto SDK
(shameless plug)

- Java SDK for rapid prototyping of cryptographic protocols
- API follows concepts from theoretical crypto
- Currently implements all building-blocks of Phish-Market
  - Generic implementation of El-Gamal, Pedersen
  - Instantiations over elliptic curves and over $\mathbb{Z}_p^*$
  - Automatic Fiat-Shamir converter for $\sum$-Protocols

- Get Qilin: http://qilin.seas.harvard.edu/
Open Questions

• Solve related data-sharing problems?
  – Much easier if we don’t need to handle previously known URLs

• Implement generic secure computation to prevent tag leaks

• Side-channels?
Thank You