GNUnet-BOSS: A multiplicative secret sharing subsystem for GNUnet

Julius Bünger & Raphael Arias

Technische Universität München
Course: Peer-2-Peer Systems & Security

June 12, 2014
Overview

1. Problem & motivation
Overview

1. Problem & motivation

2. Related work
Overview

1. Problem & motivation
2. Related work
3. The project
Overview

1. Problem & motivation
2. Related work
3. The project
4. Conclusion
Secret sharing

Sharing a secret
Secret sharing

Sharing a secret

- does not mean telling someone the secret but instead
Secret sharing

Sharing a secret

- does *not* mean telling someone the secret but instead
- splitting a secret up into parts that can be distributed and
Secret sharing

Sharing a secret
- does *not* mean telling someone the secret but instead
- splitting a secret up into parts that can be distributed and
- (when enough parts (*shares, shadows*) come together) reconstructed
A practical example

You and a group of friends/coworkers discover a magic number that lets you efficiently factor large numbers (or solve discrete logarithm, etc.).

- Too dangerous to publish, as most of crypto would break.
You and a group of friends/colleagues discover a magic number that lets you efficiently factor large numbers (or solve discrete logarithm, etc.).

- Too dangerous to publish, as most of crypto would break.
- You don’t want to destroy the magic number (maybe you will some day need it).
A practical example

You and a group of friends/colleagues discover a magic number that lets you efficiently factor large numbers (or solve discrete logarithm, etc.).

- Too dangerous to publish, as most of crypto would break.
- You don’t want to destroy the magic number (maybe you will some day need it).
- Eliminate the risk of one of you publishing on their own.
A practical example

You and a group of friends/coworkers discover a magic number that lets you efficiently factor large numbers (or solve discrete logarithm, etc.).

- Too dangerous to publish, as most of crypto would break.
- You don’t want to destroy the magic number (maybe you will some day need it).
- Eliminate the risk of one of you publishing on your own.
- So you decide to
A practical example

You and a group of friends/colleagues discover a magic number that lets you efficiently factor large numbers (or solve discrete logarithm, etc.).

- Too dangerous to publish, as most of crypto would break.
- You don’t want to destroy the magic number (maybe you will some day need it).
- Eliminate the risk of one of you publishing on their own.
- So you decide to
  - split the number so that each of you owns a share
A practical example

You and a group of friends/colleagues discover a magic number that lets you efficiently factor large numbers (or solve discrete logarithm, etc.).

- Too dangerous to publish, as most of crypto would break.
- You don’t want to destroy the magic number (maybe you will some day need it).
- Eliminate the risk of one of you publishing on their own.
- So you decide to
  - split the number so that each of you owns a share but only all of you together are able to reconstruct it.
You and a group of friends/colleagues discover a magic number that lets you efficiently factor large numbers (or solve discrete logarithm, etc.).

- Too dangerous to publish, as most of crypto would break.
- You don’t want to destroy the magic number (maybe you will some day need it).
- Eliminate the risk of one of you publishing on their own.
- So you decide to
  - split the number so that each of you owns a share but only all of you together are able to reconstruct it.
  - destroy the original secret.
Operations on shared secrets

- Some secret sharing schemes allow operations on shared secrets.
Operations on shared secrets

- Some secret sharing schemes allow operations on shared secrets.
- You can add two secrets that are shared between multiple parties and obtain a shared secret of the sum.
Operations on shared secrets

- Some secret sharing schemes allow operations on shared secrets.
- You can add two secrets that are shared between multiple parties and obtain a shared secret of the sum.
- You can scale a shared secret (multiply by a scalar value).
Operations on shared secrets

• Some secret sharing schemes allow operations on shared secrets.
• You can add two secrets that are shared between multiple parties and obtain a shared secret of the sum.
• You can scale a shared secret (multiply by a scalar value).
• And, if you are really courageous you can multiply shared secrets (with each other).
Threshold schemes

- In threshold secret sharing schemes, the number of shares created can differ from the number of shares required to recover the secret.
In threshold secret sharing schemes, the number of shares created can differ from the number of shares required to recover the secret.

$n$ shares are distributed, but $t + 1$ suffice to recover the secret (while $t$ shares do not reveal it).
Threshold schemes

- In threshold secret sharing schemes, the number of shares created can differ from the number of shares required to recover the secret.
- \( n \) shares are distributed, but \( t + 1 \) suffice to recover the secret (while \( t \) shares do not reveal it).
- Call this a \((t + 1, n)\)-threshold scheme.
Applications (Du, Atallah) [8]

- General
Applications (Du, Atallah) [8]

- General

- Multiplicative
Applications (Du, Atallah) [8]

- General
  - Remote database access / private information retrieval (PIR) [6]

- Multiplicative
Applications (Du, Atallah) [8]

- General
  - Remote database access / private information retrieval (PIR) [6]
  - Distributed key generation, decryption, signing

- Multiplicative
Applications (Du, Atallah) [8]

- General
  - Remote database access / private information retrieval (PIR) [6]
  - Distributed key generation, decryption, signing
  - Secure voting
- Multiplicative
Applications (Du, Atallah) [8]

- General
  - Remote database access / private information retrieval (PIR) [6]
  - Distributed key generation, decryption, signing
  - Secure voting
- Multiplicative
  - Constraint solving / linear programming [3, 10, 12]
Applications (Du, Atallah) [8]

- General
  - Remote database access / private information retrieval (PIR) [6]
  - Distributed key generation, decryption, signing
  - Secure voting

- Multiplicative
  - Constraint solving / linear programming [3, 10, 12]
  - Data mining / statistical analysis [1]
Applications (Du, Atallah) [8]

- **General**
  - Remote database access / private information retrieval (PIR) [6]
  - Distributed key generation, decryption, signing
  - Secure voting

- **Multiplicative**
  - Constraint solving / linear programming [3, 10, 12]
  - Data mining / statistical analysis [1]
  - Geometric computation [2]
Applications (Du, Atallah) [8]

- **General**
  - Remote database access / private information retrieval (PIR) [6]
  - Distributed key generation, decryption, signing
  - Secure voting

- **Multiplicative**
  - Constraint solving / linear programming [3, 10, 12]
  - Data mining / statistical analysis [1]
  - Geometric computation [2]
  - Intrusion detection [5]
Applications (Du, Atallah) [8]

- General
  - Remote database access / private information retrieval (PIR) [6]
  - Distributed key generation, decryption, signing
  - Secure voting
- Multiplicative
  - Constraint solving / linear programming [3, 10, 12]
  - Data mining / statistical analysis [1]
  - Geometric computation [2]
  - Intrusion detection [5]

Crypto-anarchy?!
Use case: secure private linear programming [7]

- Transformation-based approach to secure multi-party linear programming.
Use case: secure private linear programming [7]

- Transformation-based approach to secure multi-party linear programming.
- Linear program is shared among parties and then transformed via distributed matrix multiplications, to be solved by a standard LP solver.
Use case: secure private linear programming [7]

- Transformation-based approach to secure multi-party linear programming.
- Linear program is shared among parties and then transformed via distributed matrix multiplications, to be solved by a standard LP solver.
- There are also direct approaches (SMC version of the Simplex algorithm).
Shamir: “How to share a secret” [11]

- $(t + 1, n)$-threshold scheme.
Shamir: “How to share a secret” [11]

- $(t + 1, n)$-threshold scheme.
- “Encode” the secret as constant coefficient of a (otherwise random) polynomial of degree $t$. 

![Graph of a polynomial function]
Shamir: “How to share a secret” [11]

- \((t + 1, n)\)-threshold scheme.
- “Encode” the secret as constant coefficient of a (otherwise random) polynomial of degree \(t\).

Create \(n\) shares by evaluating polynomial at \(n\) distinct, non-zero points.
Shamir: “How to share a secret” [11]

- \((t + 1, n)\)-threshold scheme.
- “Encode” the secret as constant coefficient of a (otherwise random) polynomial of degree \(t\).

- Create \(n\) shares by evaluating polynomial at \(n\) distinct, non-zero points.
- With \(t + 1\) such shares the polynomial is uniquely identified and can be recovered and evaluated at 0 to yield the secret.
Shamir: “How to share a secret” [11]

- $(t + 1, n)$-threshold scheme.
- “Encode” the secret as constant coefficient of a (otherwise random) polynomial of degree $t$.

- Create $n$ shares by evaluating polynomial at $n$ distinct, non-zero points.
- With $t + 1$ such shares the polynomial is uniquely identified and can be recovered and evaluated at 0 to yield the secret.
- $t$ shares (or less) do not reveal anything about the secret.
Verifiable secret sharing (Feldman) [9]

- With verifiable secret sharing the goal is to ensure that the party who claims to be sharing the secret is actually handing out shares of the same secret to everyone.
Verifiable secret sharing (Feldman) [9]

- With verifiable secret sharing the goal is to ensure that the party who claims to be sharing the secret is actually handing out shares of the same secret to everyone.

- Achieved by having the sharing party commit to the shares by publishing a special value.
Verifiable secret sharing (Feldman) [9]

- With verifiable secret sharing the goal is to ensure that the party who claims to be sharing the secret is actually handing out shares of the same secret to everyone.
- Achieved by having the sharing party commit to the shares by publishing a special value.
- This is already implemented in GNUnet in the context of anonymous voting / distributed key generation / cooperative decryption. (Thank you, Florian!) – Remains to be seen how adaptable it is for our purpose.
Ben-Or, Goldwasser & Wigderson [4]

- Adding secrets: adding shares means adding polynomials.
  Adding polynomials means the constant coefficients are added.
Ben-Or, Goldwasser & Wigderson [4]

- Adding secrets: adding shares means adding polynomials. Adding polynomials means the constant coefficients are added.
- The same holds for scaling.
Ben-Or, Goldwasser & Wigderson [4]

- Adding secrets: adding shares means adding polynomials. Adding polynomials means the constant coefficients are added.
- The same holds for scaling.
- Actually, it also holds for multiplication. But there is a problem. Any ideas?
Ben-Or, Goldwasser & Wigderson [4]

- Adding secrets: adding shares means adding polynomials. Adding polynomials means the constant coefficients are added.
- The same holds for scaling.
- Actually, it also holds for multiplication. But there is a problem. Any ideas?
- Multiplying polynomials increases the degree! We need to be able to reconstruct the secret from the polynomial, so choose \( t = \left\lfloor \frac{n-1}{2} \right\rfloor \).
Project goal

- A GNUnet service that allows sharing secrets, as well as performing additions, scaling and multiplication of shared secrets.
A GNUnet service that allows sharing secrets, as well as performing additions, scaling and multiplication of shared secrets.

If secrets should not be reconstructed immediately after multiplication (because they are intermediate results) and still used in further computations (multiplications), the degree needs to be reduced.
Project goal

- A GNUnet service that allows sharing secrets, as well as performing additions, scaling and multiplication of shared secrets.

- If secrets should not be reconstructed immediately after multiplication (because they are intermediate results) and still used in further computations (multiplications), the degree needs to be reduced.

- This can be done using a rather complicated and communication-heavy protocol.
Project goal

- A GNUnet service that allows sharing secrets, as well as performing additions, scaling and multiplication of shared secrets.
- If secrets should not be reconstructed immediately after multiplication (because they are intermediate results) and still used in further computations (multiplications), the degree needs to be reduced.
- This can be done using a rather complicated and communication-heavy protocol.
- Our goal: make it (almost) transparent to the user of the GNUnet service by wrapping it in a simple API.
Design of the library

- Standard GNUnet Tier-architecture: client connects to service.
Design of the library

- Standard GNUnet Tier-architecture: client connects to service.
- One client needs to initiate a secret sharing session. This is communicated to the service.
Design of the library

- Standard GNUnet Tier-architecture: client connects to service.
- One client needs to initiate a secret sharing session. This is communicated to the service.
- The service initiates communication with other participating peers.
Design of the library

- Standard GNUnet Tier-architecture: client connects to service.
- One client needs to initiate a secret sharing session. This is communicated to the service.
- The service initiates communication with other participating peers.
- Assumption: The initiator has a list of potential participants which he/she passes to the service.
Design of the library

- Standard GNUnet Tier-architecture: client connects to service.
- One client needs to initiate a secret sharing session. This is communicated to the service.
- The service initiates communication with other participating peers.
- Assumption: The initiator has a list of potential participants which he/she passes to the service.
- A few (very general) message types INIT, INFORM, REQUEST.
Design of the library

- Standard GNUnet Tier-architecture: client connects to service.
- One client needs to initiate a secret sharing session. This is communicated to the service.
- The service initiates communication with other participating peers.
- Assumption: The initiator has a list of potential participants which he/she passes to the service.
- A few (very general) message types INIT, INFORM, REQUEST.
- Keep protocol extensible by allowing message subtypes (initialize session / sharing / operation, etc.).
Project milestones

- API definition (✓)
Project milestones

- API definition (√)
- GNUnet-boss (Ben-Or Secret Sharing) service skeleton + P2P/IPC protocols (√)
Project milestones

- API definition (√)
- GNUnet-boss (Ben-Or Secret Sharing) service skeleton + P2P/IPC protocols (√)
- core service capabilities
Project milestones

- API definition (✓)
- GNUnet-boss (Ben-Or Secret Sharing) service skeleton + P2P/IPC protocols (✓)
- core service capabilities
- command-line tools + test cases
Project milestones

- API definition (✓)
- GNUnet-boss (Ben-Or Secret Sharing) service skeleton + P2P/IPC protocols (✓)
- core service capabilities
- command-line tools + test cases
- code cleanup & refactoring
Some stretch goals or “What we probably won’t finish”

- Do verifiable secret sharing (just like Florian).
Some stretch goals or “What we probably won’t finish”

- Do verifiable secret sharing (just like Florian).
- Make some fundamental changes to the underlying secret sharing library.
Some stretch goals or “What we probably won’t finish”

- Do verifiable secret sharing (just like Florian).
- Make some fundamental changes to the underlying secret sharing library.
- Adjust to allow applications to perform (specifically for the problem) optimized operations (e.g. matrix multiplication).
Some stretch goals or “What we probably won’t finish”

- Do verifiable secret sharing (just like Florian).
- Make some fundamental changes to the underlying secret sharing library.
- Adjust to allow applications to perform (specifically for the problem) optimized operations (e.g. matrix multiplication).
- Allow transference of shares.
Problems to be aware of

- How should client disconnects be handled? What about “partial” disconnects (only some nodes)?
Problems to be aware of

- How should client disconnects be handled? What about “partial” disconnects (only some nodes)?
- Communication complexity issues.
Problems to be aware of

- How should client disconnects be handled? What about “partial” disconnects (only some nodes)?
- Communication complexity issues.
- Reconstruction requires trust. If I send you my share, can I be sure you will send me yours?
Problems to be aware of

- How should client disconnects be handled? What about “partial” disconnects (only some nodes)?
- Communication complexity issues.
- Reconstruction requires trust. If I send you my share, can I be sure you will send me yours?
- All other problems we are too unimaginative to think of. Any ideas?
That’s it!

Thank you for your attention!

Questions, comments?
The complete degree reduction protocol

Let \( f[i] \) denote the share of the polynomial \( f \) at point \( x_i \) (owned by player \( i \)). Let \( h(x) \) be the polynomial for which degree reduction should be performed.

1. \( \forall i. \) Player \( P_i \) generates random polynomial \( q_i \) with secret 0 and \( \forall j. \) shares \( q_i[j] \) with player \( P_j \).
2. \( \forall i, j. \) Player \( P_i \) sums up \( q_j[i] \) he received in the last step to obtain \( q[i] \).
3. \( \forall i. \) Player \( P_i \) computes \( \beta_i = h'[i] = h[i] + q[i] \).
4. \( \forall i, j. \) Player \( P_i \) shares \( \beta_i[j] \) with player \( P_j \).
5. \( \forall i, j. \) Player \( P_i \) computes \( \delta_j[i] = A \cdot \beta[i] \) and restores \( \delta_j[i] \) to player \( P_j \).
6. \( \forall j. \) Player \( P_j \) computes \( \delta_j \), his share of the reduced polynomial.


Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation.


Jannik Dreier and Florian Kerschbaum.
Practical privacy-preserving multiparty linear programming based on problem transformation.

Wenliang Du and Mikhail J. Atallah.
Secure multi-party computation problems and their applications: A review and open problems.

Paul Feldman.
A practical scheme for non-interactive verifiable secret sharing.
