Cryptography And Voting

Panagiotis Grontas

$\mu\Pi\lambda\forall$

14/3/2013

Overview

- Motivation
- Cryptography Building Blocks
- Mixnets: A solution to voter privacy
- Verifiably Correct Mixnets
- Almost Verifiably Correct Mixnets
- Conclusion

Motivation

Election Systems: High Level properties

- **Integrity**
  - Votes are cast as intended
  - Votes are counted as cast
- **Ballot Secrecy** Nobody can figure out how you voted (privacy), even if you try to prove it (selling/coercion).
- **Authentication and Authorisation**
  - Only authorised voters can vote
  - a specified number of times as stated in election law
- **Enfranchisement** All voters must have the opportunity and be encouraged to vote
- **Availability** (Voting,Tallying)
- **Efficiency** (Cost,Time)

Remark: Authentication vs Secrecy vs Enfranchisement

Election Systems: Interpretation of properties in building a voting system

- **Privacy** The votes must remain secret
- **Individual Verifiability** Each voter can check that their ballot was included in the outcome (*to ensure integrity*)
- **Universal Verifiability** All voters can check that a voter's ballot was included in the outcome (*to ensure integrity*)
- **Receipt - Freeness** A voter cannot prove how she voted even if she wants to.
- **Robustness** Nobody can disrupt an election (*to ensure availability*)
- **Fairness** No partial results are known. No vote cancellation/duplication. (*to ensure availability, integrity, privacy*)

Remark: Individual Verifiability vs Receipt - Freeness

Electronic Voting For Better Elections(?) I

- Traditional Systems lack many of the properties we described earlier
  - Lack of Individual/Universal Verifiability (we cast our votes, without verifying that they are counted)
  - Trust is based on tradition and conflict of interest
- By computerising the elections we can actually improve the voting process
  - By counting faster
  - By enabling winner selection by a variety of social choice functions
  - Most importantly: by enabling some of the before mentioned properties
  - We can design the election system, from the ground up following specifications
- But computers themselves introduce many problems
  - Can we implement systems with conflicting characteristics?
  - Lack Of Transparency
  - eVoting is like voting by proxy. Can we trust another entity to vote for us?
  - For an example: Check the HBO documentary Hacking Democracy!

Electronic Voting For Better Elections(?) II

- One Solution: Open Source Voting Software
  - Open Source Code can be scrutinised by competing parties and everybody else
  - Voters can build the tallying programs themselves
  - Will surely play a role in the future of voting
  - **But:** How can we be sure of the actual bits that do the tallying?
- **The Solution: Cryptography**
  - Cryptography has been used to keep secrets
  - Cryptography can be used to build trust
  - How: By keeping secrecy and providing verification of each operation

Cryptography Building Blocks

Hash Functions

A function h that maps arbitrary size data (message) to fixed size data (hash) with the following properties

- Given the message it is easy to calculate the hash
- Given the hash it is computationally infeasible to find the message
- Given a message m it is computationally infeasible to find another message m' such that $h(m) = h(m')$
- It is computationally infeasible to find two messages m_1, m_2 such that $h(m_1) = h(m_2)$

Public Key Cryptosystems

- Enable exchange of secret messages without prior engagements
- Introduced by Diffie And Hellman in 1976
- Each user has a public and a private key
- In order to send an encrypted message
  - The public key is retrieved
  - The message is encrypted with the it
  - Upon receipt, the message is decrypted with the private key
- Three algorithms are needed (Key Generation, Encryption, Decryption)
- Security based on (conjectured) hard problems from number theory (factoring, discrete log, quadratic residuosity)
- Can be turned around to provide signatures (encrypt with the private key)

RSA Encryption (1977) I

- **Generate keys**
  - Select Randomly and Independently Two Large Primes p, q
  - Calculate product $n = p \cdot q$
  - Calculate $\phi(n) = (p-1) \cdot (q-1)$
  - Randomly select $e \in \mathbb{Z}_n^*$ st: $gcd(e, \phi(n)) = 1$
  - Calculate reverse $d = e^{-1} mod \phi(n)$
  - Public key is (e, n) and private key is (p, q, d)
- **Encrypt** Message m : Raise to the public key $c = m^e mod n$
- **Decrypt** message c : Raise to the private key $c^d mod n = m^{ed} mod n = m$
- For security: randomize encryption. Append random padding to the message.
- if n can be factored than breaking RSA is easy
- Threshold decryption
  - Break the private key into n pieces so that that t parties can decrypt it
  - Enables the distribution of trust

Homomorphic Encryption

- Computation with encrypted data.
- $E(m_1) \otimes E(m_2) = E(m1 \oplus m2)$
- Apply a function to the ciphertexts that corresponds to another function on the plaintexts. The result can be obtained by one decryption.
- For simple tallying we would require to evaluate a function on ciphertexts that corresponds to adding the plaintexts (additive homomorphism)
- For other social choice functions we would require computation of arbitrary functions on encrypted data.
- It can be done ... in theory (Gentry, 2010)

ElGamal Encryption (1984)

- Randomised Public Key Encryption From Diffie-Hellman Key Exchange
- Key Generation
  - Select 2 large primes p, q st $q \mid (p-1)$ and a generator g
  - Randomly select $x \in_R \mathbb{Z}_q$
  - Calculate $y = g^x modp$
  - Return $(pk = y, sk = x)$
- Encrypt Message m: Multiply with randomisation of public key
  - Randomly select $r \in_R \mathbb{Z}_q$
  - Calculate $G = g^r\, modp$
  - Calculate $M = m \cdot y^r\, modp$
  - Return $c = (G, M)$
- Decrypt message (G, M) with secret key x
  - return $M/G^x\, modp$
- Security based on difficulty of computing discrete logs

Useful Elgamal properties I

- **Reencryption**
  - Change ciphertext without affecting decryption

$$ReEnc(c, r') = c \cdot Enc(1, r') = (g^{r+r'}, m \cdot (g^x)^{r+r'})$$

  - No knowledge of secret key is required.
  - Without the secret key or the re randomisation factor it is infeasible to show that two messages are reencryptions of each other.
- **Multiplicative Homomorphism**
  - Let m_1, m_2 plaintexts. Then $Enc(m_1) \cdot Enc(m_2) = Enc(m_1 \cdot m_2)$
  - In elections we would desire **additive** homomorphism
  - ElGamal Solution: encrypt message m as $(G, M) = (g^r mod p, g^m \cdot y mod p)$
  - Problem:Need to solve DLOG, too many exponentiations
  - Other cryptosystems provide additive homomorphism

Commitments

- Commitment Schemes
  - Commit to a value
  - without revealing it (*hiding* property)
  - and without being able to change it (*binding* property)
- An application: Coin flipping over the telephone
  - Alice and Bob are in different locations but want to flip a coin
  - Alice select head/tails
  - Bob flips the coin
  - Bob doesn't have to flip the coin, he can just announce that he wins

Solution

- Commit to heads or tails
- Flip the coin and announce the result
- Reveal the commitment
- Check the result

Zero Knowledge Proofs (Goldwasser, Micali, Rackoff - 1985) I

- Interactive protocol between 2 parties (prover, verifier)
- **Objective:** The prover wants to convince the verifier about the knowledge of a secret, without disclosing (any part) of it
- Properties:
  - Completeness: Honest prover convinces honest verifier with overwhelming probability
  - Soundness: Dishonest prover cannot succeed with overwhelming probability
  - Zero Knowledge: The verifier cannot learn anything from the protocol

Zero Knowledge Proofs (Goldwasser, Micali, Rackoff - 1985) II

An illustrating example

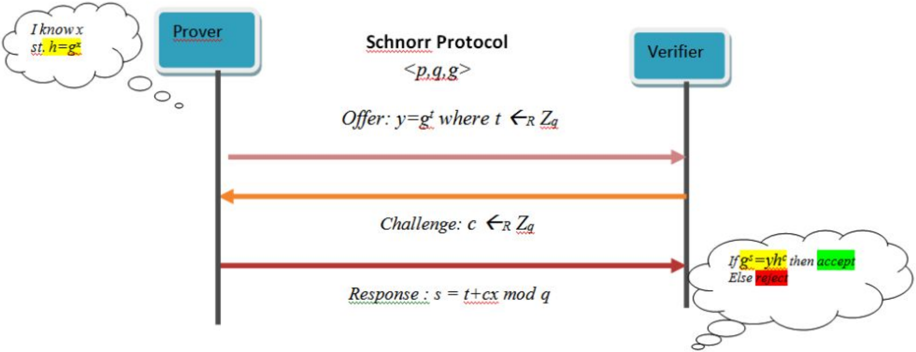
- **Prover** holds two *identical* boxes of different color
- **Verifier** is color blind
- **Prover** wants to convince the Verifier that the boxes have different color

Zero Knowledge Proofs (Goldwasser, Micali, Rackoff - 1985) III

The protocol

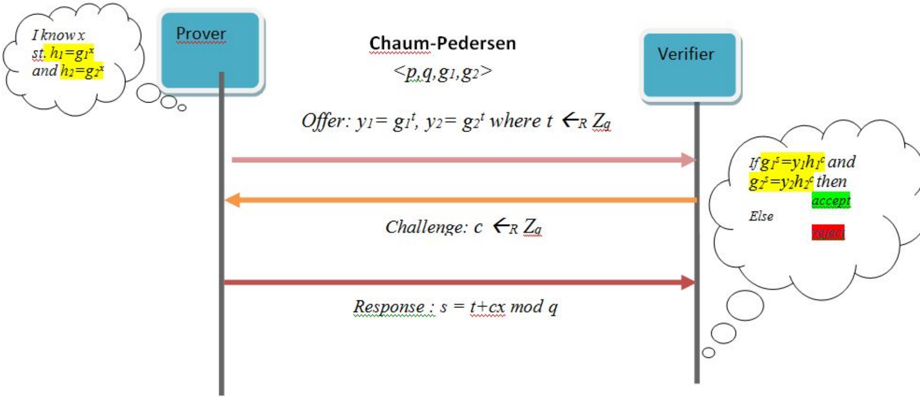
1. **Prover** gives the boxes to the verifier
2. **Verifier** hides the boxes behind her back, one box per hand
3. With probability $\frac{1}{2}$ verifier switches boxes in each hand, behind her back
4. **Verifier** reveals boxes
5. **Prover** can tell whether the verifier switched hands
6. Repeat n times to decrease cheating probability to $\frac{1}{2^n}$
7. **Verifier** is convinced, that the boxes have different color, without ever knowing what it is

Prove that you know DLOG (Schnorr, 1991)



Authentication using Zero Knowledge Proofs: Prove that you know the password, without revealing it.

Prove DLOG equality (Chaum - Pedersen protocol, 1992)



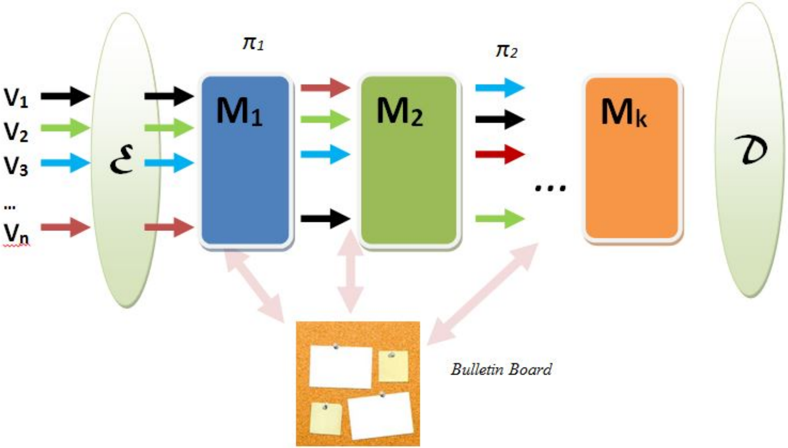
You can Prove Yourself (Fiat - Shamir heuristic, 1987) Replace verifier with hash function

Mixnets

Overview

- A solution to voter privacy - Main idea by David Chaum (1981)
- Generic primitive for anonymous channel (the original paper has 3500 citations!)
- Has been used for anonymous email, anonymous browsing, private payment system, multiparty computation and of course *elections*)
- A mixnet consists of a number of mix servers that are operated by different (mistrusting) parties
- Input: The messages to be anonymised
- Output: A permutation of the input
- To achieve anonymity: No single output item, must match an input item
- An input item is hidden by encryption and shuffling
- In theory: an honest mix server suffices to achieve privacy

Mixnets



Voting With Mixnets:Main Idea

- Voters create their ballots B_i
- Initial Encryption $C_{i0} = Enc(B_i)$
- Encrypted Ballots enter the mixnet
- Each mix server permutes and *changes* the encrypted items $C_{\pi_j(i)j} = X_j(C_{ij-1})$
- After all mixing has occured the **unencrypted ballots** are posted to the bulletin board
- The social choice function is computed
- All communication is achieved by reading from and appending to the bulletin board

Decryption Mixnets

- The original Chaum idea
- Encrypt the ballot with the public key of the mix servers in reverse order
- $C_{i0} = E_{pk1} \circ E_{pk2} \circ \ldots \circ E_{pkt}(B_i)$
- Each mix server peels of a layer of encryption (decrypts with its secret key) and performs the shuffle
- After the final stage, all ballots are decrypted
- **Remarks:**
  - Independent of underlying crypto system
  - The ciphertext size is proportional to the number of mix servers
  - A mix server can block the mixing process by refusing to decrypt
  - The last mix server knows the votes and can block the elections (share the final decryption key)

ReEncryption Mixnets - (Park, Itoh, Kurosawa 1993)

- Version 1
  - Each mix server re randomises the ballots by reencryption
  - A final decryption stage is needed
  - The decryption key must be shared to various parties
  - The decryption is jointly done by all mix servers.
- Version 2
  - Combines decryption and reencryption
  - Each mix server partially decrypts the ballots by applying its secret key.
  - Then re randomises by reencryption
  - The last mix server decrypts

A basic attack (Pfitzmann, 1995) I

Active Attack: Trace an encrypted input by injecting a correlated message

Track input m_i for participant P_i

- Initial Encryption $c_{i0} = (g^R, m_i \cdot (y_1, \cdots, y_k)^R)$
- Mix server j input: $c_{ij} = (g^{R'}, m_i \cdot (y_j, \cdots, y_k)^{R'}) = (t, u)$
- For some random x generate $c''_{ij} = (t^x, u^x) = (g^{R'x}, m_i^x \cdot (y_j, \cdots, y_k)^{R'x})$
- Output will contain both m_i^x, m_i
- Raise all output messages to the x and check for duplicates.

Reminder: El Gamal has multiplicative homomorphism $(E(m)^x = E(m^x))$

A basic attack (Pfitzmann, 1995) II

Track input $m_1, \ldots, m_s$ for participants $P_i, \ldots, P_s$

- Choose random values $x_1, \ldots, x_s$
- Calculate $c = \prod_{i=1}^{s} c_{ij}^{x_i}$
- Inject or replace with c
- Output will contain the decryption $m*$ of c
- Look for s messages such that $m* = \prod_{i=1}^{s} m_i^{x_i}$

A basic attack (Pfitzmann, 1995) III

Remarks

- Applies to both decryption and reencryption mixnets
- If there is a check on number of input items a colluding participant's message must be omitted
- Solution: Redundancy in messages in order to detect the attack
  - Increases Ciphertext Size
  - Does not work if the last mix server is corrupt, since it can replace the m_i^x with a correct looking message after the message correlation
- Something stronger is needed

Problems

Problems

- At the initial encryption stage, a different vote might be encrypted (vote changing, vote copying, vote cancelling)
- A mix server can change some of the input votes, by replacing them in the output.
- A subset of the mix servers might cooperate to break anonymity by tracing messages

Solutions must be efficient, correct and privacy respecting

Solutions

Main Idea

- Zero - knowledge proof of the contents of the vote
- Zero - knowledge proof of the correctness of the shuffle

- Correctness of initial encryption $\Leftrightarrow$ Prove that you know DLOG
- Correctness of Shuffling $\Leftrightarrow$
  Prove that each encrypted vote in the input, appears in the output in a valid reencrypted form

Verifiable Mixnets

Killian-Sako Verifiable Mixnet (1995) I

- The first universally verifiable mixnet
- Elgamal Reencryption based Mixnet
- Verification based on cut and choose protocol

Cut - And - Choose Proof Of Correct Reencryption

- Prove knowledge of secret key and partial decryption.
- Perform secondary reencryption with new randomisation factor
- Reveal secondary reencryption or difference between primary and secondary reencryption

Killian-Sako Verifiable Mixnet (1995) II

Cut - And - Choose Proof Of Correct Shuffle - Main idea

- Each mix server generates another permutation and randomisation values
- Perform reencryption and shuffling according to them (secondary shuffle)
- Reveal secondary shuffle or the difference between primary and secondary shuffle

Mixnets based on permutation networks - 1999 I

- Each mix server M_i simulates a sorting network.
- Reencrypts then sorts the inputs
- The mix server is built in a bottom up by combining smaller comparator functions
- **Proof Of Correctness:** Prove for a 2×2 sorting network using the Chaum-Pedersen Protocol 4 times and generalise for $n \times n$ sorting network

Furukawa and Sako Mixing - 2001 I

- Mixing is represented as matrix multiplication.
- Permutation matrix

$$A_{ij} = \begin{cases} 1, \pi(i) = j \\ 0, \text{otherwise} \end{cases}$$

- For example the permutation $\pi(1,2,3) = (2,3,1)$ can be represented using the matrix:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

- Shuffling and re encryption can be represented using a permutation matrix.
- Prove that r_i and $[A_{ij}]$ exists based on the key observation that a matrix $[A_{ij}]$ is a permutation matrix iff the dot product of two columns is 0 (if they are different) and 1 (if they are the same)

Neff Verifiable Mixnet (2001, 2003) I

- Mix inputs and outputs are represented as polynomials P_{in}, P_{out}
- Key Property: A polynomial is unaffected by permutation of the roots. $\prod_{i=1}^{n}(m_i - x) = \prod_{i=1}^{n}(m_{\pi(i)} - x)$
- Verifier chooses a random $t \in Z_q$
- Evaluates both input and output polynomials
- The results match with very high probability

Verifiable Mixnets: Performance

Cost = Total Number of Exponentiations For

- Initial Encryption
- Proof of Reencryption and Shuffling
- Decryption

Performance for n voters and k mix servers:

- **Sako - Killian** $642nk$
- **Sorting Networks** $7nlogn(2k-1)$
- **Furukawa Sako** $18n(2k-1)$
- **Neff** $8n(2k-1)$

In practice: Neff Mixnet: 10^6 votes $\Rightarrow 20$ hours to mix and verify.

Lesson:Zero Knowledge Proofs Are Computationally Expensive.

Almost Verifiable Mixnets

Randomised Partial Checking I

- RPC - Jakobsson, Juels, Rivest - 2001
- Create an efficient verifiable mixnet out of any cryptosystem
- **Idea**: Give up the expensive notion of **proof**
- Provide **strong evidence** that the mix server has operated correctly (ie. the output is a processed permutation of the input)
- Strong Evidence = Probabilistic Verification
- Partial Revelation of the input/output correspondence.
- For n items, choose randomly $\frac{n}{2}$ and reveal the input/output relation.
- The mix server has no control of which items are revealed.
- **Objective:** A cheater cannot get away with altering too many votes
- **Tradeoff:** Privacy and Correctness

Randomised Partial Checking II

Operation

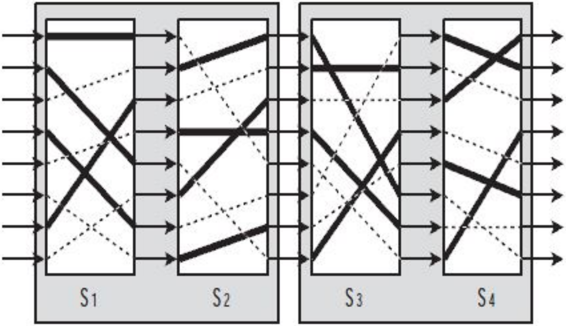
- X_j is a cryptographic operation that transforms ciphertext c to c' (reencryption, decryption)
- Mix server M_j randomly selects a permutation π_j
- Commit to the permutation by publishing to the bulletin board
  - A commitment Γ_j^{in} that input i maps to output $\pi_j(i)$, if j is odd
  - A commitment Γ_j^{out} that output i came from $\pi_j^{-1}(i)$, if j is even
- Proof of correct operation: **Partial** Revelation
  - Reveal information that allows anyone to verify that $c_{ij} = X_j(c_{kj-1})$
  - What to reveal: randomness, i, k
  - Also reveal the commitments
- Verifier validates the transformation

Randomised Partial Checking III

What about privacy

- Main idea:Pair the servers so that we never reveal the same correspondence twice.
- Only the inter-pair correspondences are revealed
- Let j odd and (M_j, M_{j+1}) a server pair . Then
  - $P_{IN}(Q_j, k) = false$
  - $P_{OUT}(Q_j, i) = true$ with probability $\frac{1}{2}$
  - $P_{IN}(Q_{j+1}, i) = 1 - P_{OUT}(Q_j, i)$
  - $P_{IN}(Q_{j+1}, m) = false$
- At least one honest pair is needed for privacy

Randomised Partial Checking IV



Almost Entirely Correct Mixing (Boneh, Golle - 2002) I

- **Idea**: Sacrifice the validation of correctness for speed.
- An almost correct proof of mixing might suffice, if the margin of victory is high

Method

- Select a random subset S of mix server inputs
- Calculate the product π_s
- Reveal S to the mix server.
- Ask to product a set of outputs S' st. $\pi_s = \pi_{s'}$
- Honest mix server: Simply apply the permutation
- Cheating mix server: **Might** be impossible to find such S'.

Optimistic Mixing (Golle, Zhong, Boneh, Jakobsson, Juels - 2002) I

- Concept based on almost entirely correct mixing
- Fast proof when all mix servers are honest
- Reminder: Product preservation might not imply absence of cheating
- Cheating might be discovered, albeit after the fact. Privacy would be exposed.
- Solution: Augment with cryptographic checksums and check both product and checksums
- If a cheating mix server is found then:
  - No output is produced
  - A correct proof is executed (Neff, Furukawa-Sako)
  - Privacy is not compromised
- **Details:**User input is encrypted twice!

Optimistic Mixing (Golle, Zhong, Boneh, Jakobsson, Juels - 2002) II

- Encrypt the vote
- Hash the encryption components
- Encrypt the hash.

- Mixing proceeds as usual with permutation and reencryption
- Verification: Decrypt once and check products and checksums
- If verification succeeds then everything is OK. Decrypt the vote and tally.
- If cheating is discovered then the cause of cheating is sought for the triples that do not checksum.
- How:
  - Starting from the end each server reveals (input, randomisation) for the triples in question.
  - If the first server is reached than the mix worked correctly.

Optimistic Mixing (Golle, Zhong, Boneh, Jakobsson, Juels - 2002) III

- Cheating was due to the users. Solution: Ignore the cheating senders and count the vote.
- If a server is exposed as cheating then repeat with slower and verifiable mixnet.

- Cheating will not expose privacy, since the cheating server will occur on (single) ciphertexts

Conclusion

Conclusion

- Electronic voting **can** improve the voting process
- If implemented correctly, it allows for properties that are not found even in the traditional systems that we have grown to trust
- Cryptography can help rebuild that trust, by allowing for secrecy and verification
- Mixnets are a mature technology that has been extensively researched in the last 20 years and can be used to anonymise the votes
- There exist many protocols for mixnets that combine efficiency, correctness, privacy to some extent
- Many eVoting systems rely on mixnets (Pret A′ Voter, Verificatum)
- The building blocks are there but much work must be done in their composition, implementation and proof of security
- The goal however remains: 'Trust nothing but verify everything' (*Ben Adida - creator of Helios*)

Thank you!

References

References I

Masayuki Abe.
Universally verifiable mix-net with verification work independent of the number of mix-servers.
In Kaisa Nyberg, editor, *Advances in Cryptology ▯ EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 437–447. Springer Berlin Heidelberg, 1998.

B. Adida.
Advances in cryptographic voting systems.
PhD thesis, Massachusetts Institute of Technology, 2006.

Ben Adida.
Verifying elections with cryptography, 2007.

Dan Boneh and Philippe Golle.
Almost entirely correct mixing with applications to voting.
In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 68–77, New York, NY, USA, 2002. ACM.

David Chaum.
Untraceable electronic mail, return addresses, and digital pseudonyms.
Commun. ACM, 24(2):84–88, 1981.

Jun Furukawa and Kazue Sako.
An efficient scheme for proving a shuffle.
In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 368–387, London, UK, UK, 2001. Springer-Verlag.

References II

Philippe Golle, Sheng Zhong, Dan Boneh, Markus Jakobsson, and Ari Juels.
Optimistic mixing for exit-polls.
In *Asiacrypt 2002, LNCS 2501*, pages 451–465. Springer-Verlag, 2002.

J. Alex Halderman.
Securing digital democracy.
Coursera Online Course, September 2012.

Markus Jakobsson.
Flash mixing.
In *PODC '99: Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, pages 83–89, New York, NY, USA, 1999. ACM.

Markus Jakobsson, Ari Juels, and Ronald L. Rivest.
Making mix nets robust for electronic voting by randomized partial checking.
In *In USENIX Security Symposium*, pages 339–353, 2002.

Aggelos Kiayias.
Cryptography primitives and protocols.
`crypto.di.uoa.gr/class/Kryptographia/Semeioseis_files/Cryptograph_Primitives_and_Protocols.pdf`, 2011.

Joe Kilian and Kazue Sako.
Receipt-free MIX-type voting scheme - a practical solution to the implementation of a voting booth.
In *Proceedings of EUROCRYPT 1995*. Springer-Verlag, 1995.

Shahram Khazaei and Douglas Wikström.
Randomized partial checking revisited.
Technical report, Cryptology ePrint Archive, Report 2012/063, 2 012. http://eprint. iacr. org, 2012.

References III

Jakobsson Markus and Juels Ari.
Millimix: Mixing in small batches.
Technical report, 1999.

M. Michels and P. Horster.
Some remarks on a receipt-free and universally verifiable mix-type voting scheme.
In *Advances in Cryptology□ASIACRYPT'96*, pages 125–132. Springer, 1996.

C. Andrew Neff.
A verifiable secret shuffle and its application to e-voting.
In *Proceedings of the 8th ACM conference on Computer and Communications Security*, CCS '01, pages 116–125, New York, NY, USA, 2001. ACM.

C. Andrew Neff.
Verifiable mixing (shuffling) of elgamal pairs.
Technical report, In proceedings of PET '03, LNCS series, 2004.

Ryan O'Donnell.
An example of zero knowledge proof.
`mathoverflow.net/questions/22624/example-of-a-good-zero-knowledge-proof`, 2011.

B. Pfitzmann.
Breaking an efficient anonymous channel.
In *Advances in Cryptology□EUROCRYPT'94*, pages 332–340. Springer, 1995.

Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa.
Efficient anonymous channel and all/nothing election scheme.
In *EUROCRYPT*, pages 248–259, 1993.

References IV

B. Pfitzmann and A. Pfitzmann.
How to break the direct rsa-implementation of mixes.
In *Advances in Cryptology□EUROCRYPT□89*, pages 373–381. Springer, 1990.

Ronald Rivest.
6.897 - selected topics in cryptography, 2004.

Bruce Schneier.
Applied cryptography (2nd ed.): protocols, algorithms, and source code in C.
John Wiley & Sons, Inc., New York, NY, USA, 1995.

Gerardo I. Simari.
A primer on zero knowledge protocols.
`cs.uns.edu.ar/~gis/publications/zkp-simari2002.pdf`, 2002.

Douglas Wikström.
Five practical attacks for □optimistic mixing for exit-polls□.
In *Selected Areas in Cryptography*, pages 160–174. Springer, 2004.

Douglas Winkstom.
Mixnets for voting.
Secvote 2010, 2010.