

Performance Evaluation of Secure Network Coding using Homomorphic Signatures

NETCOD 2011, Beijing July 25-25,2011

SeungHoon Lee, Mario Gerla (UCLA)

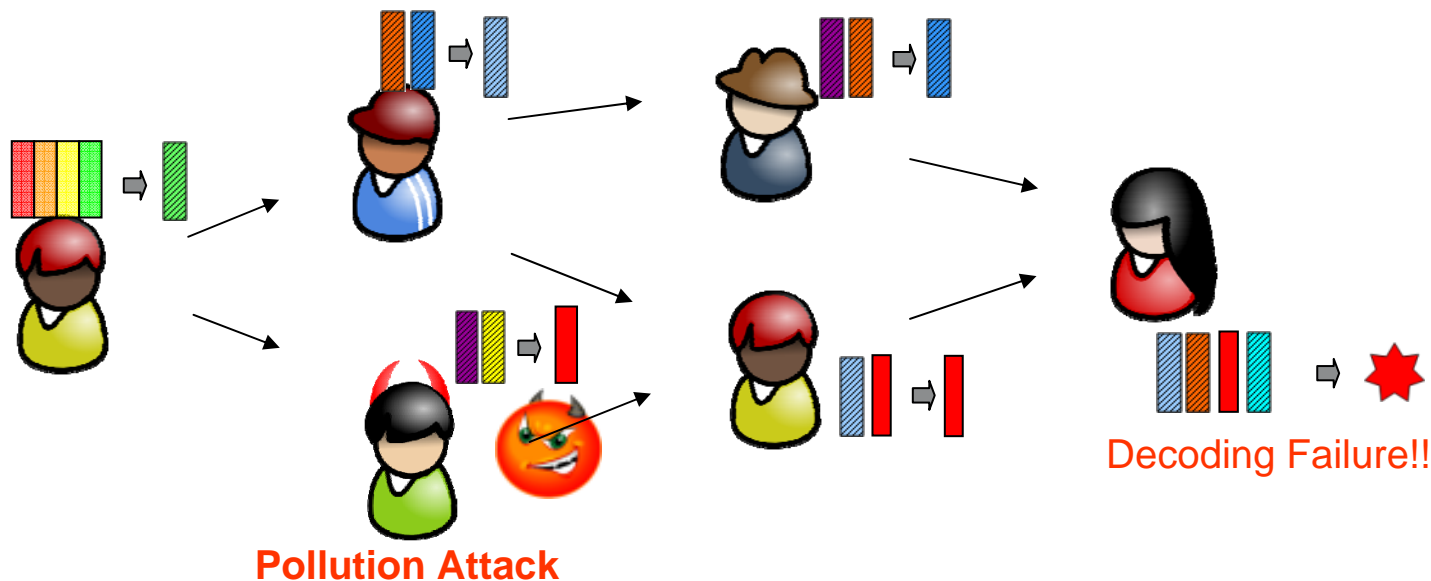
Hugo Krawczyk, Kang-Won Lee (IBM Research)

Elizabeth A. Quaglia (Royal Holloway, University of London)



Challenge of **Network Coding** in Wireless Networks

- Vulnerable to *pollution attacks*
 - Nature of **packet mixing**
 - One invalid packet can destroy the entire generation
 - Attackers inject corrupted packets
 - No pollution detection/prevention mechanism in conventional network coding



Related Works (2) - Theory

- **Homomorphic Hashing**

- [Gennaro et al, 10], [Krohn et al, 04] [Zhao et al, 07]

- **Homomorphic Signature**

- [Gennaro et al, 10], [Boneh et al, 09], [Charles et al, 06] [Johnson et al, 02]

Secure Network Coding – the Theory

- Main Result: Homomorphic hash functions/signatures preserve property under linear combinations
- Homomorphic Hashing
 - [Gennaro et al, 10], [Krohn et al, 04] [Zhao et al, 07]
 - Computationally more **efficient**,
but **requires** transmission/pre-distribution of **per generation signature**
- Homomorphic Signature
 - [Gennaro et al, 10], [Boneh et al, 09], [Charles et al, 06] [Johnson et al, 02]
 - Computationally more **expensive**,
but does **NOT** require transmission of per generation signature

Related Works - Theory vs. Practice

- Homomorphic Hashing & Signature
 - Will work, but have high **computational overhead**
 - Exponentiation, multiplication, modular operations with large integers

$$\text{Nsig}(w) = \left(\prod_{i=1}^m h_i^{u_i} \prod_{j=1}^n g_j^{v_j} \right)^d \pmod N \quad \sigma^e \stackrel{?}{=} \prod_{i=1}^m h_i^{u_i} \prod_{j=1}^n g_j^{v_j} \pmod N. \quad \sigma = \prod_{i=1}^{\ell} \sigma_i^{\alpha_i} \pmod N$$

Objective of this work

- Investigate practical aspects of Secure NC
 - Address feasibility of Secure NC in various platforms such as PCs, smartphones
 - Use the scheme GKRR proposed by Gennaro (PKC '10)
 - Evaluate in a realistic framework suitable for **mobile wireless networks**

Homomorphic Signature

Source



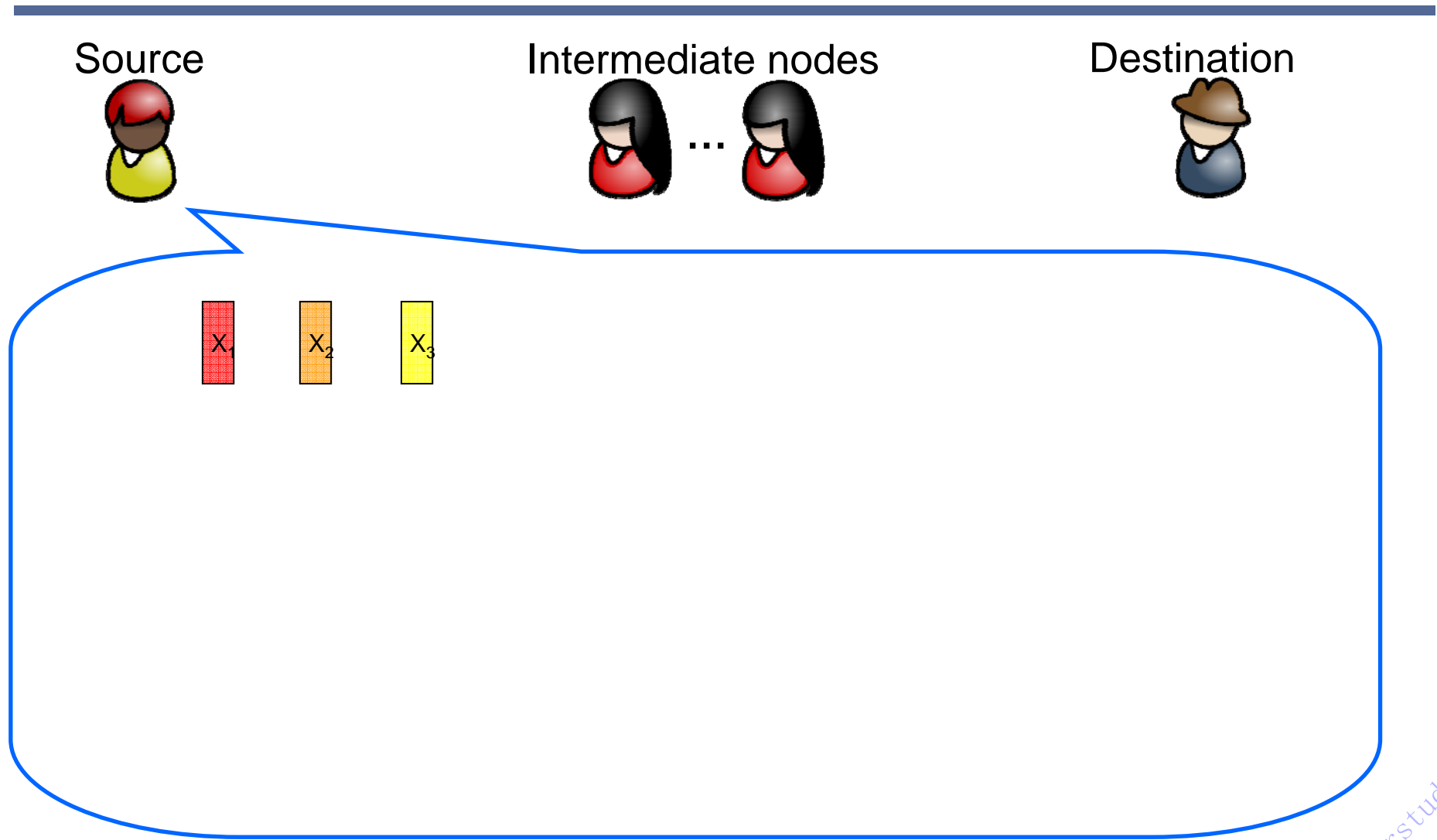
Intermediate nodes



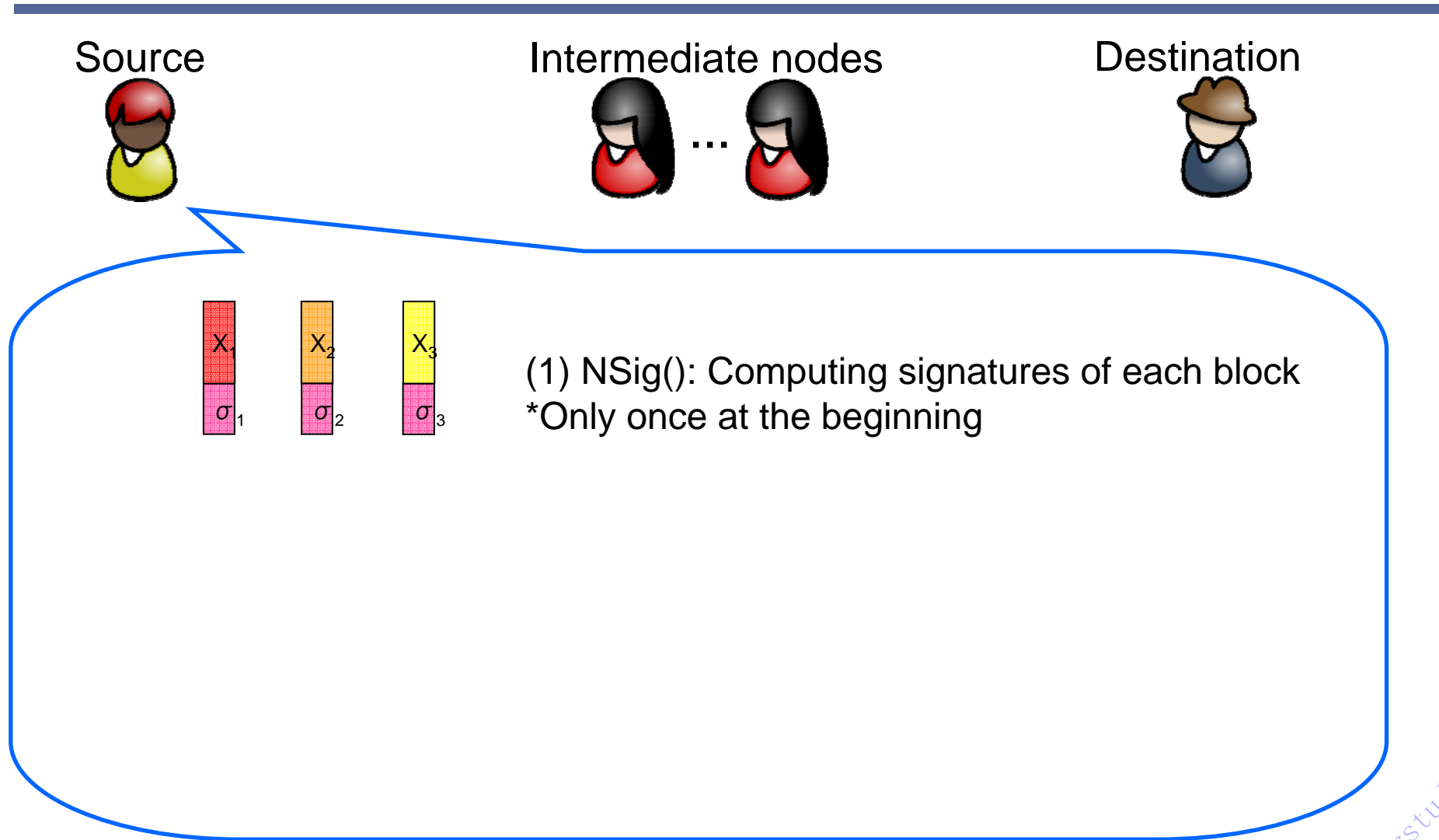
Destination



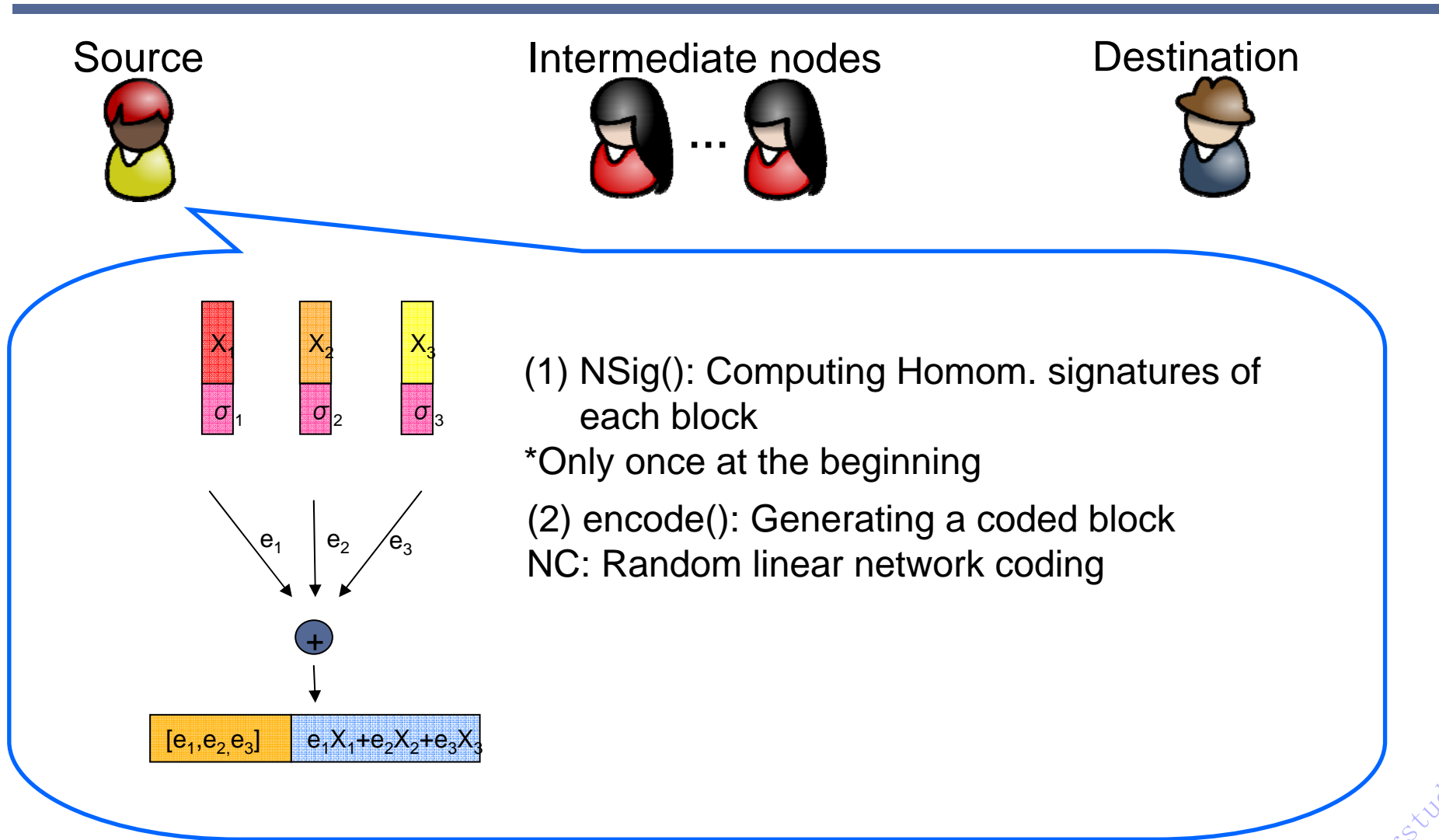
Homomorphic Signature



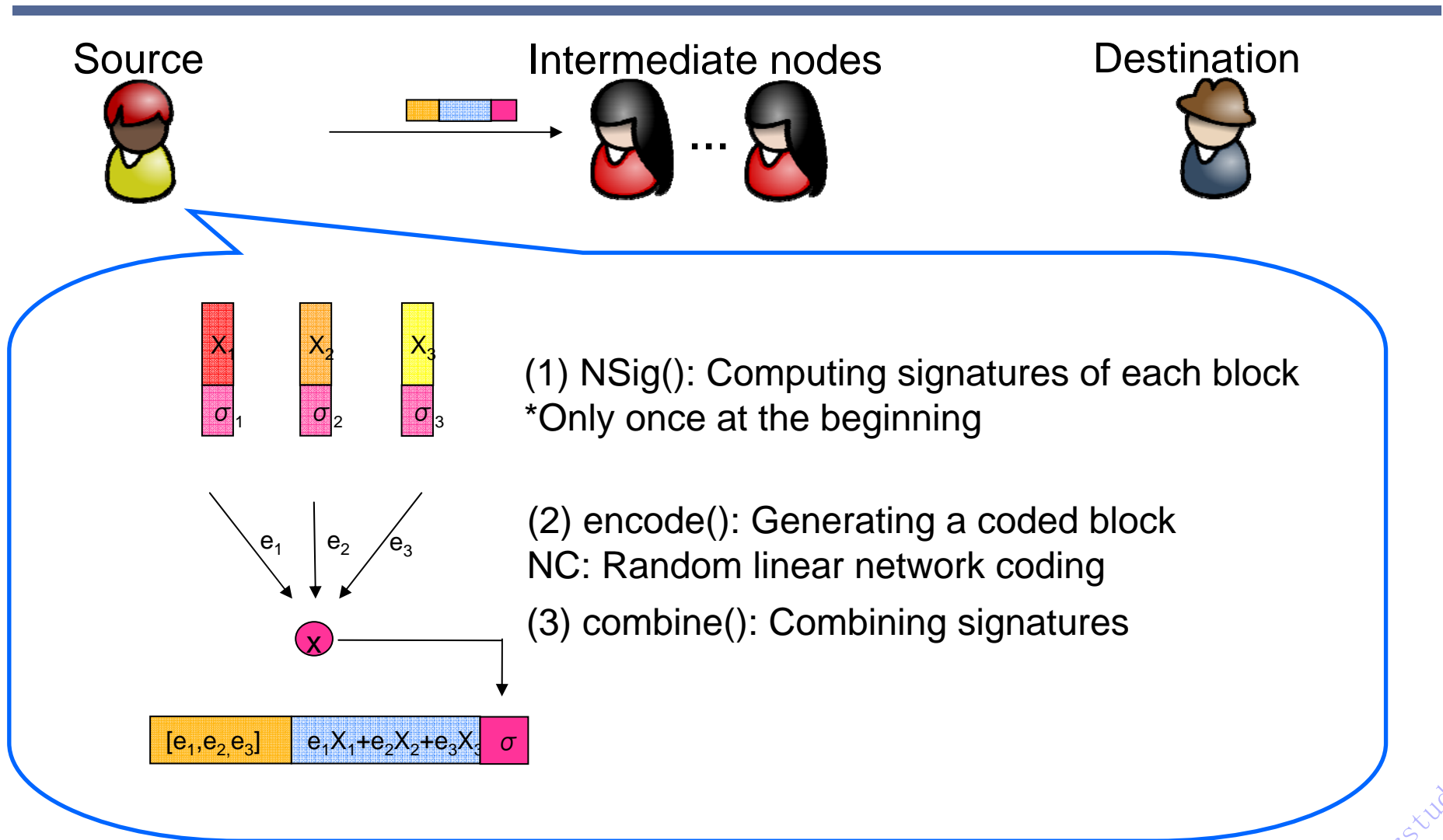
Homomorphic Signature



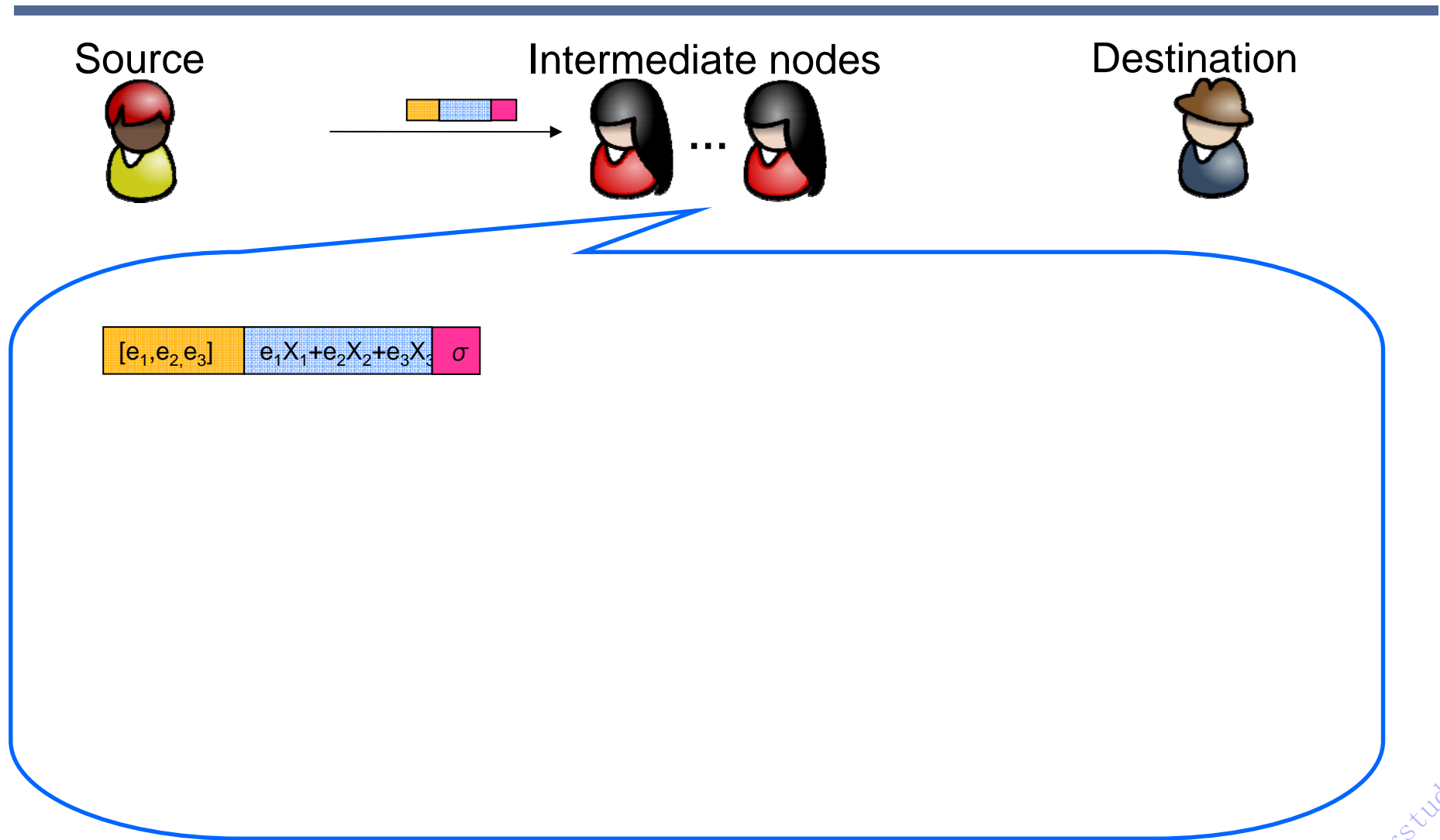
Homomorphic Signature



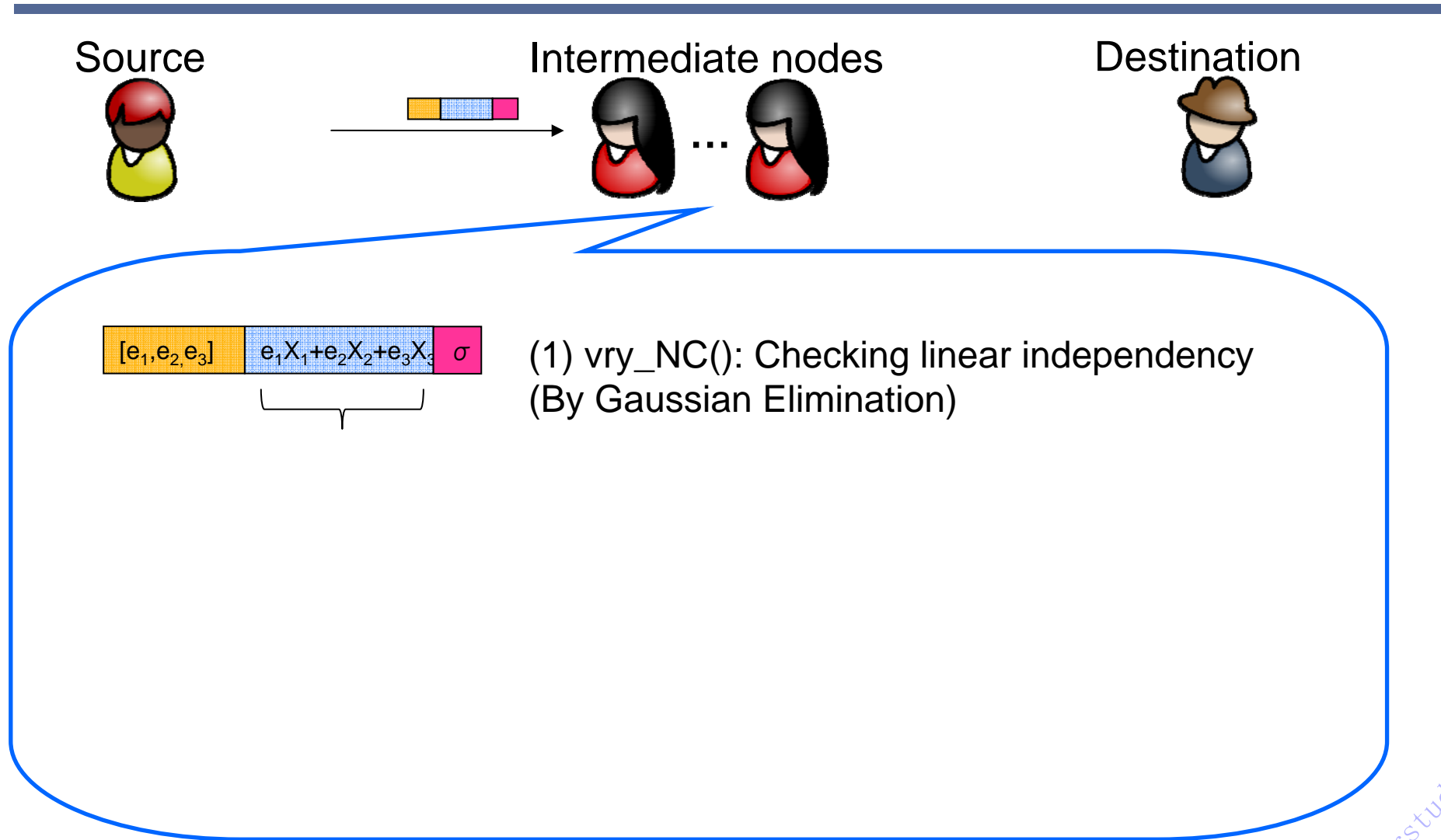
Homomorphic Signature



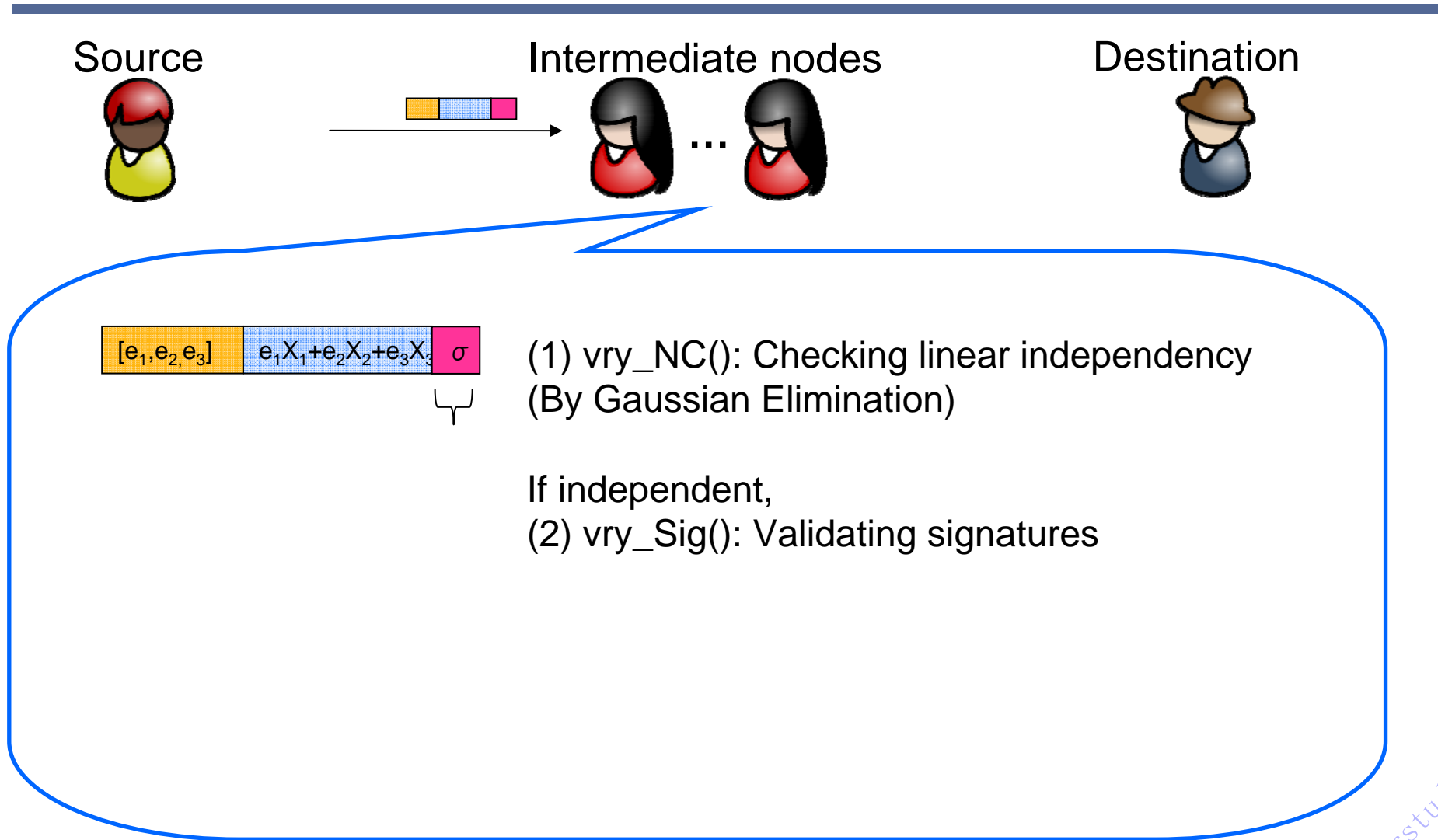
Homomorphic Signature



Homomorphic Signature



Homomorphic Signature



Homomorphic Signature

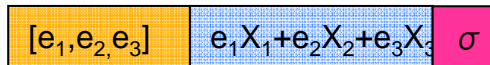
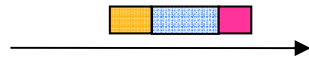
Source



Intermediate nodes



Destination



(1) vry_NC(): Checking linear independency
(By Gaussian Elimination)

If independent,

(2) vry_Sig(): Validating signatures

If valid, store the coded block

*If either verification fails, *immediately* drop.

Homomorphic Signature

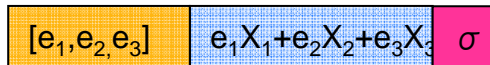
Source



Intermediate nodes



Destination



(1) `vry_NC()`: Checking linear independency
(By Gaussian Elimination)

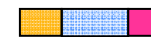
If independent,

(2) `vry_Sig()`: Validating signatures

If valid, store the coded block

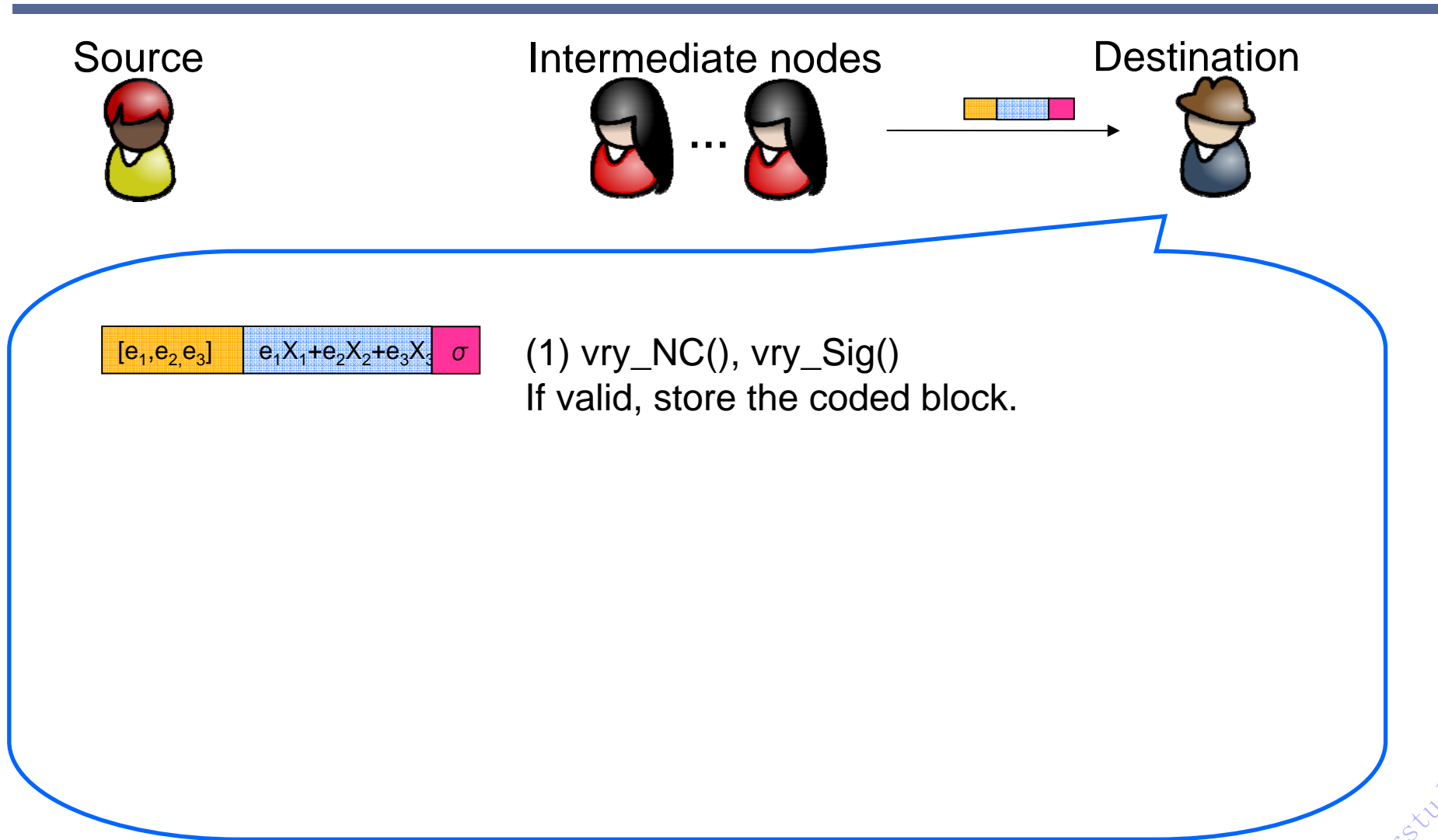
*If either verification fails, *immediately* drop.

Generate a new coded block



by `encode(data)`, `combine(signatures)`

Homomorphic Signature



Homomorphic Signature

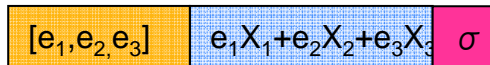
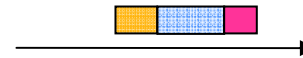
Source



Intermediate nodes



Destination

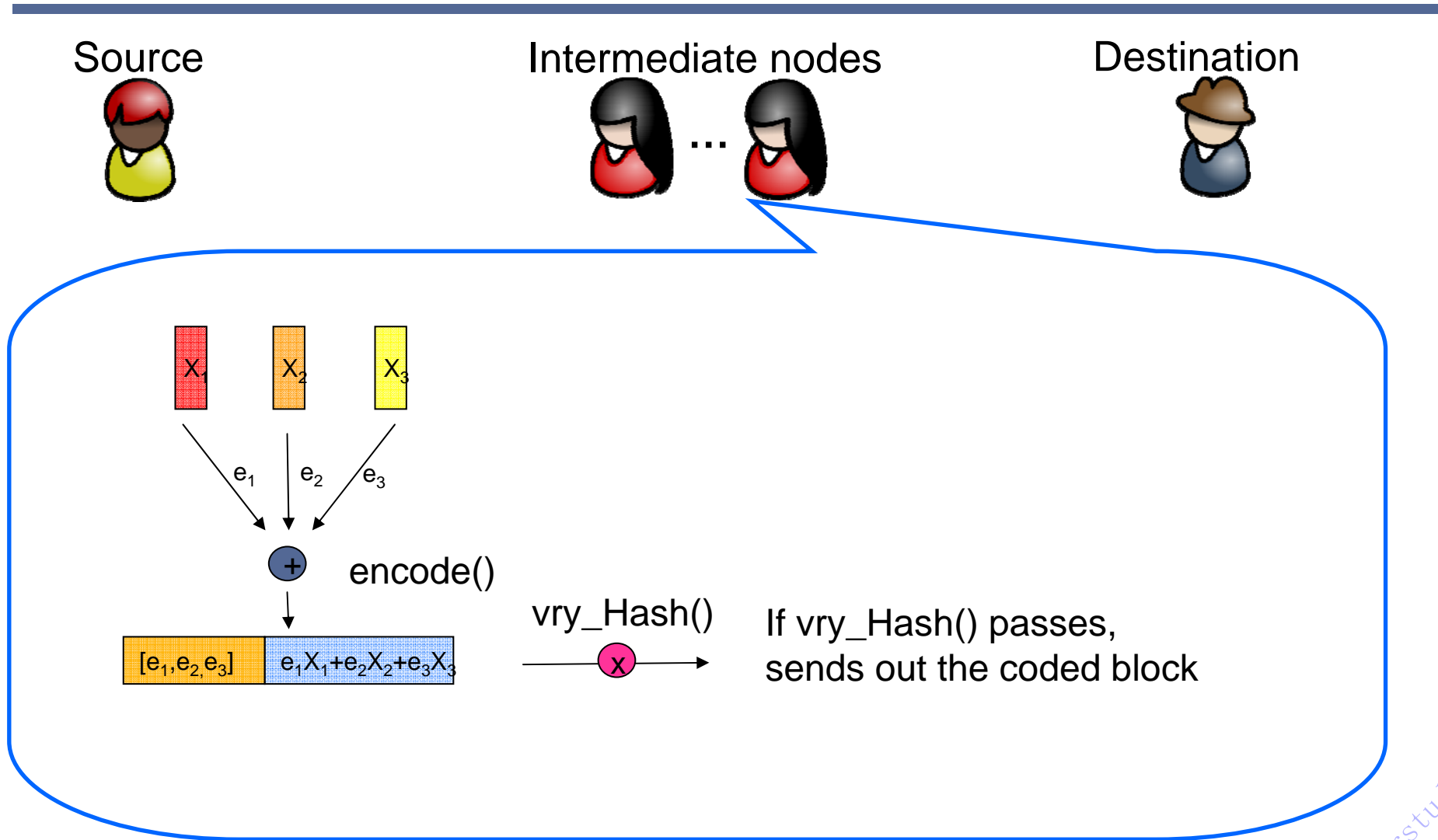


(1) `vry_NC()`, `vry_Sig()`
If valid, store the coded block.

Once collect m blocks (valid & independent),
(2) `decode()`: Recover the original data

* m : # of blocks of data in the generation

Homomorphic Hashing



Implementation & Evaluation

- Implementation on various platforms
 - Heterogeneous node types in realistic network scenarios
 - Various computational capacity (laptop vs. smartphones)

Implementation & Evaluation

- Implementation on various platforms
 - Heterogeneous node types in realistic network scenarios
 - Broad range of computational capacity (laptop vs. smartphones)

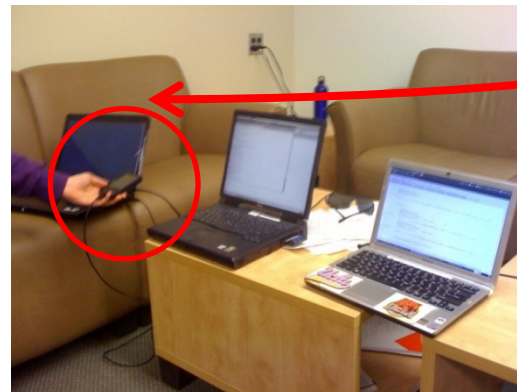
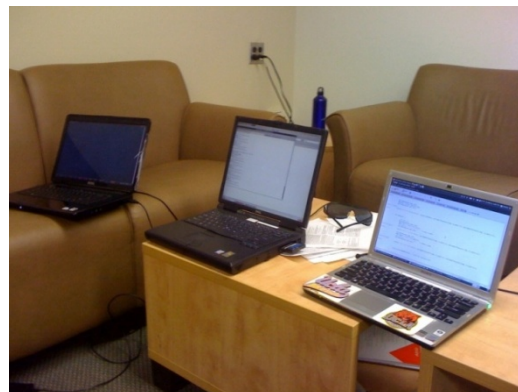
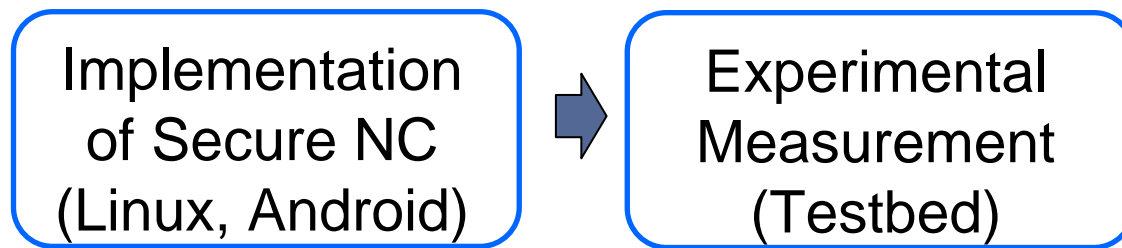
- Evaluation

Implementation
of Secure NC
(Linux, Android)

Implementation & Evaluation

- Implementation on various platforms
 - Heterogeneous node types in realistic network scenarios
 - Various computational capacity (laptop vs. smartphones)

- Evaluation

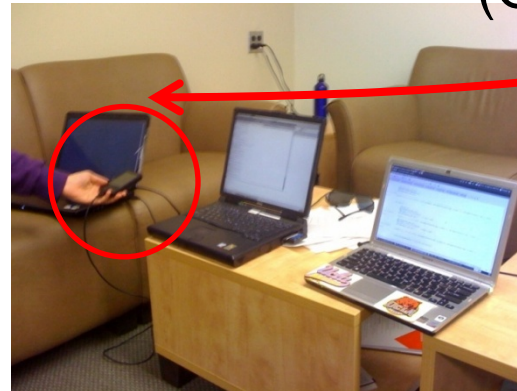
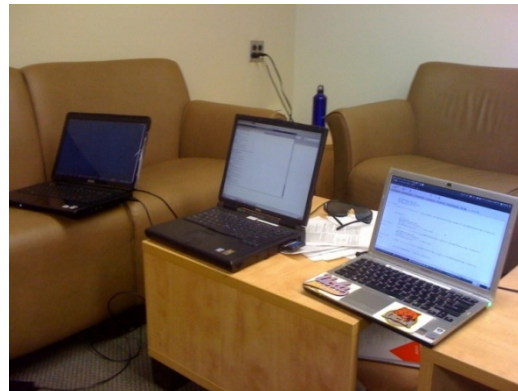
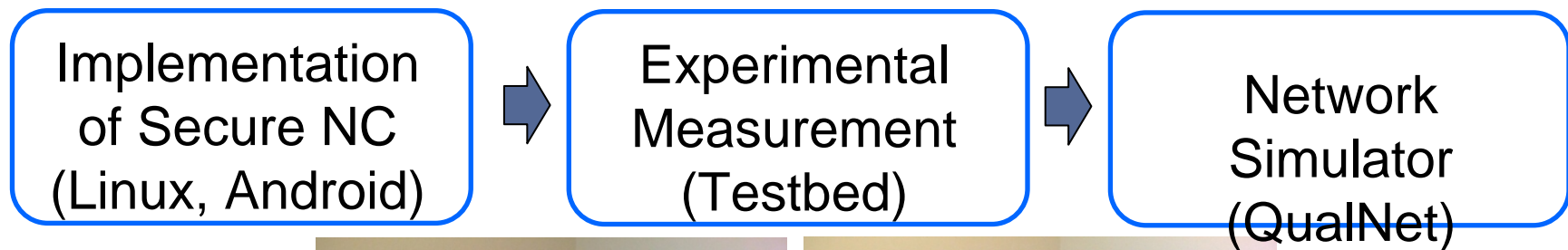


Smartphone

Implementation & Evaluation

- Implementation on various platforms
 - Heterogeneous node types in realistic network scenarios
 - Various computational capacity (laptop vs. smartphones)

■ Evaluation



Smartphone

Experimental Setup (1)

■ Hardware

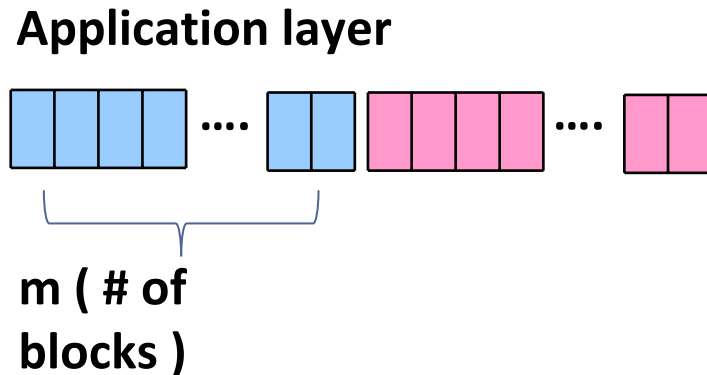
- Laptop
 - Intel Core 2 Duo T9600 processor (2.8GHz, 6MB cache)
 - RAM: 2GB
- Smartphone: T-mobile G1
 - Qualcomm MSM 7201A 528 MHz processor
 - RAM: 192MB

■ Software

- Linux platform, Android
- C++ / GMP library (for cryptography implementation)

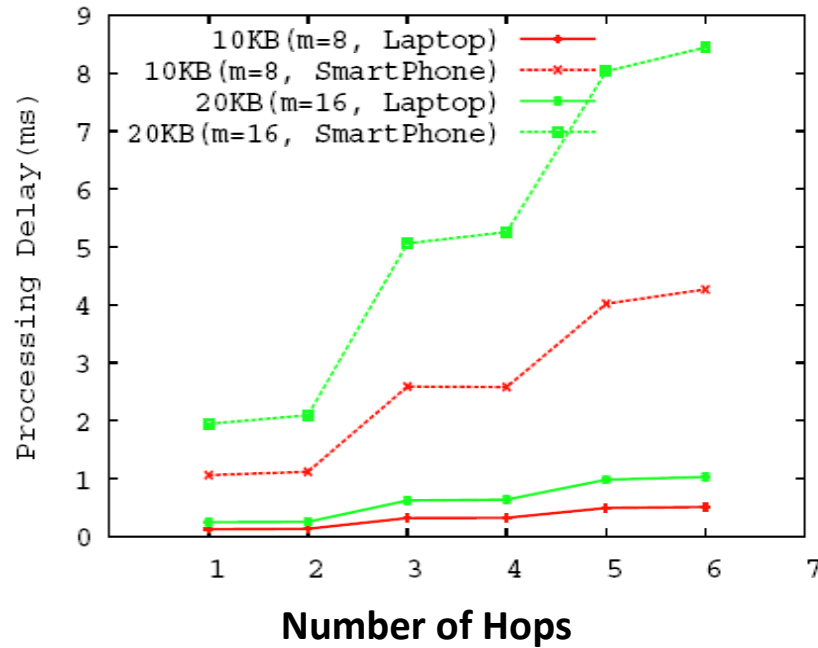
Experimental Setup (2)

- Generation size: 10KB ($m=8$), 20KB ($m=16$)

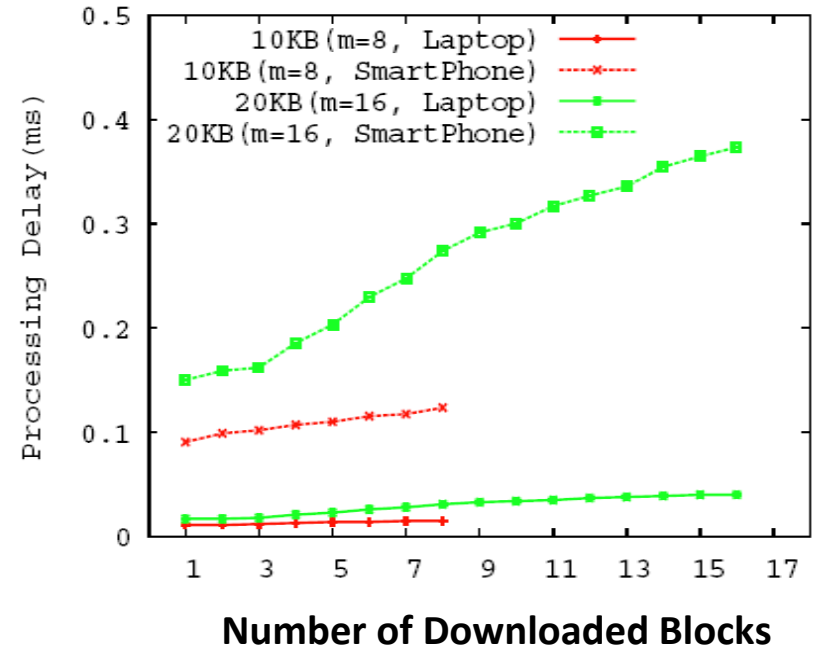


- Evaluation metrics
 - Computation delay of each operation

Experimental Results (1)



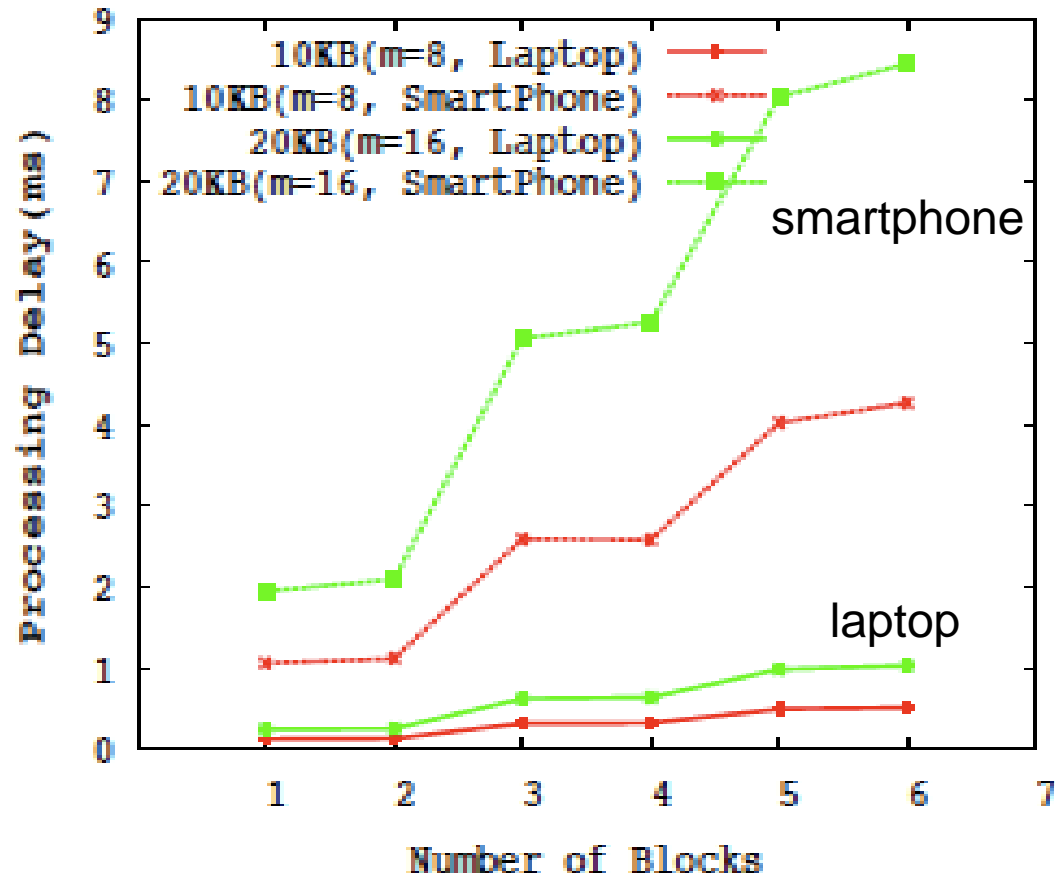
(a) Encoding Delay



(b) NC Verification Delay

- Processing delays are proportional to # of hops and # of blocks
- As downloaded more blocks, `vry_NC()` requires more delay for processing Gaussian elimination

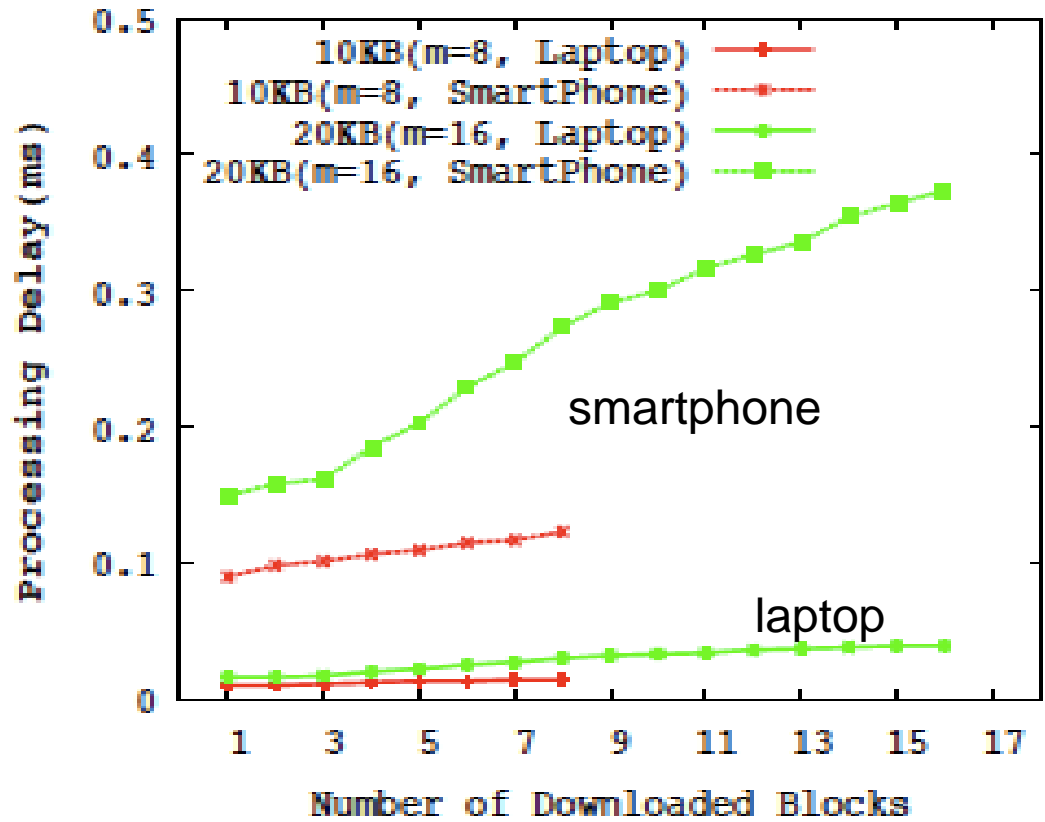
Experimental Results (1)



(a) Encoding Delay

- Processing delays are proportional to # of blocks

Experimental Results (2)



(b) NC Verification Delay

As more blocks are downloaded, `vry_NC()` requires more delay for processing Gaussian elimination

Experimental Results (3)

	10KB(m=8)	20KB(m=16)
combine()	0.16ms	0.31ms
vry_Sig()	22.17ms	22.38ms
vry_Hash()	17.44ms	17.55ms

TABLE I
PROCESSING DELAY: SECURE NETWORK CODING

Sig/Hash Verification require **100 times** more process delay than NC

Question: can we reduce Sig Verification?

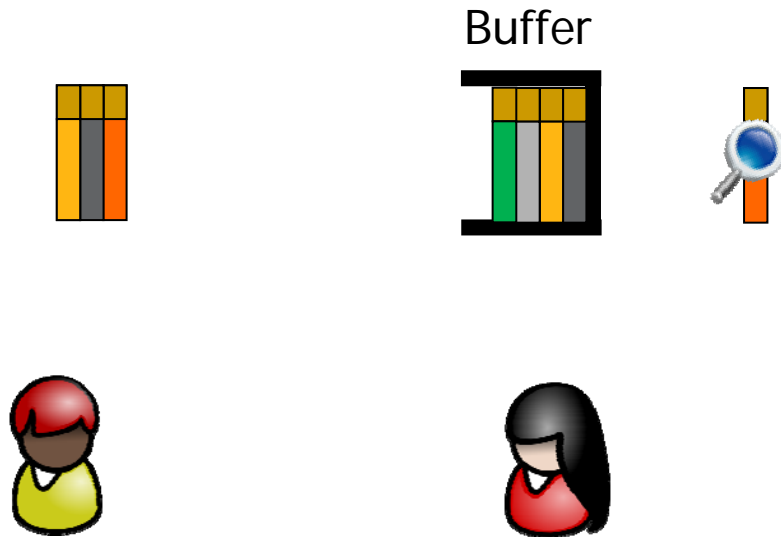
Experimental Results (2)

	10KB(m=8)	20KB(m=16)
combine()	0.16ms	0.31ms
vry_Sig()	22.17ms	22.38ms
vry_Hash()	17.44ms	17.55ms

Lessons: Major computation bottleneck
Signature/Hash verification

Batch Verification

Batch verification reduces Sig Verification O/H with no impact on security



Batch Verification

Batch verification reduces Sig Verification O/H with no impact on security

Buffer



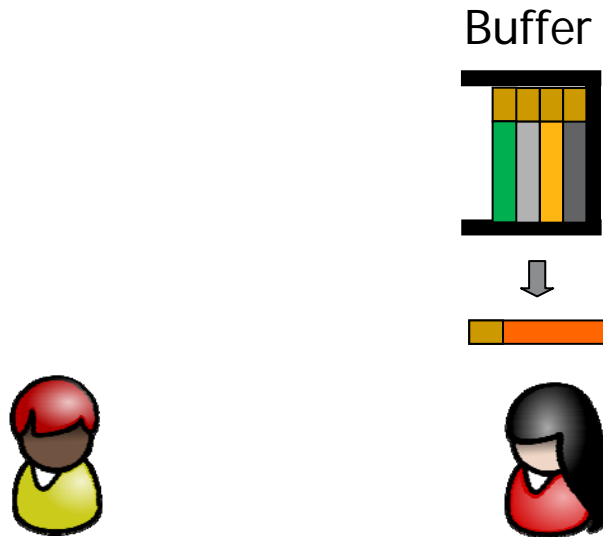
Batch Verification:

Verify multiple blocks with a single verification



Batch Verification

Batch verification reduces Sig Verification O/H with no impact on security



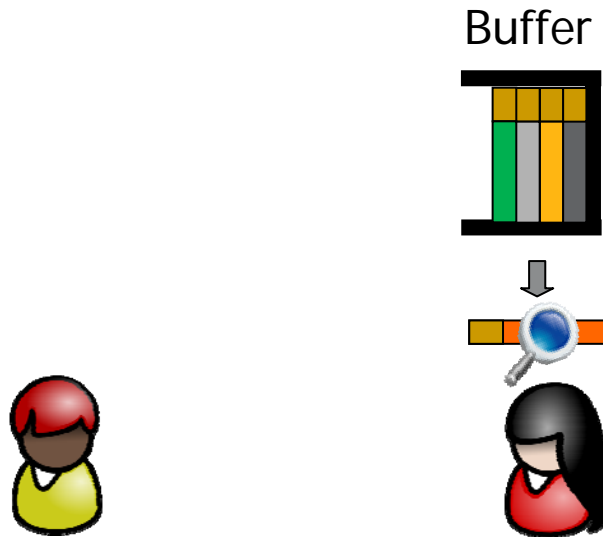
Batch Verification:

Verify multiple blocks with a single verification

1) Generate a coded block with signature

Batch Verification

Batch verification reduces Sig Verification O/H with no impact on security



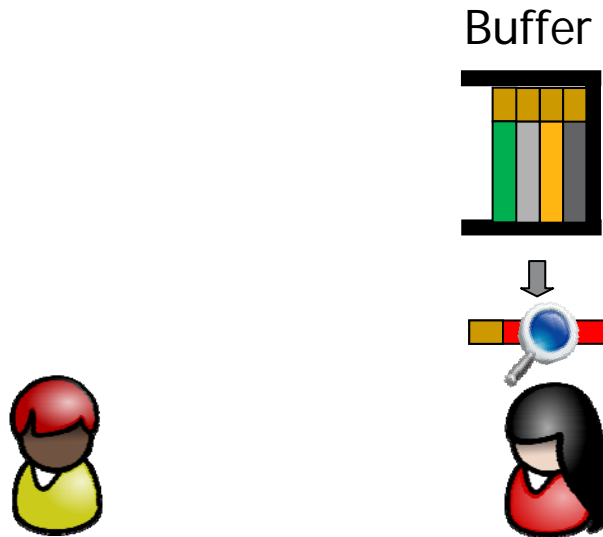
Batch Verification:

Verify multiple blocks with a single verification

- 1) Generate a coded block with signature
- 2) Verify the block
 - Valid? All of blocks in buffer Valid

Batch Verification

Batch verification reduces Sig Verification O/H with no impact on security



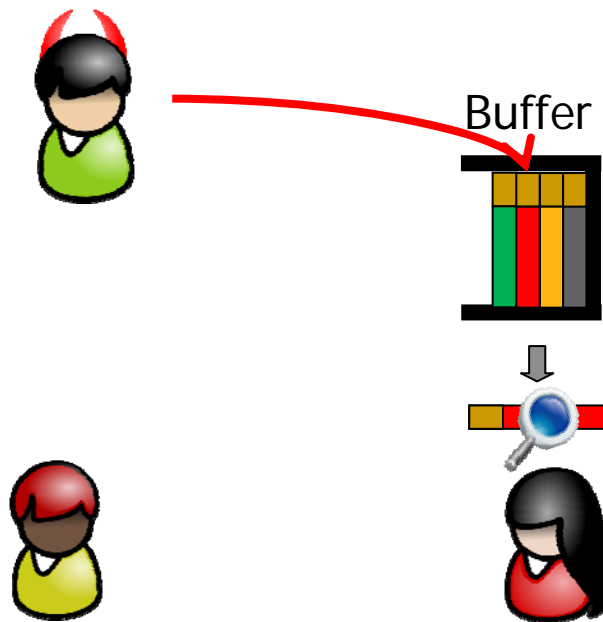
Batch Verification:

Verify multiple blocks with a single verification

- 1) Generate a coded block with signature
- 2) Verify the block
 - Valid? All of blocks in buffer Valid
 - **Invalid?** Linear or Binary Search

Batch Verification

Batch verification reduces Sig Verification O/H with no impact on security



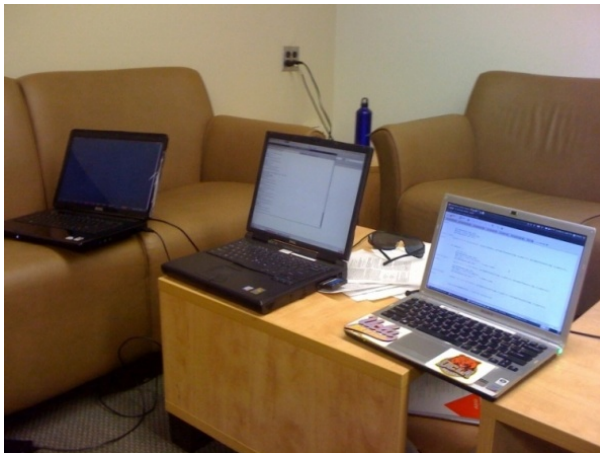
Batch Verification:

Verify multiple blocks with a single verification

- 1) Generate a coded block with signature
- 2) Verify the block
 - Valid? All of blocks in buffer Valid
 - **Invalid?** Linear or Binary Search

Batch Verification: Experiments

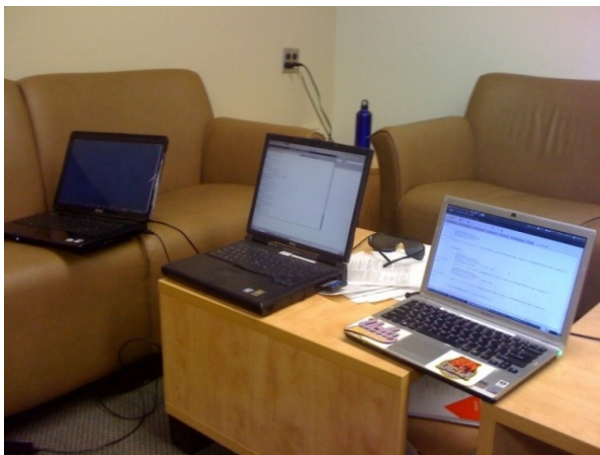
- Impact of batch verification



4 nodes (Line topology)
Data Rate: 256Kbps

Batch Verification: Experiments

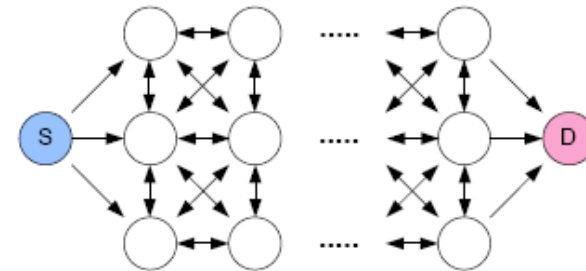
- Impact of batch verification
 - The average number of blocks in the buffer
 - 3-4 blocks
 - End-to-End delay
 - w/o Batch verification: 1.0239 sec
 - w/ Batch verification: 0.711 sec



4 nodes (Line topology)
Data Rate: 256Kbps

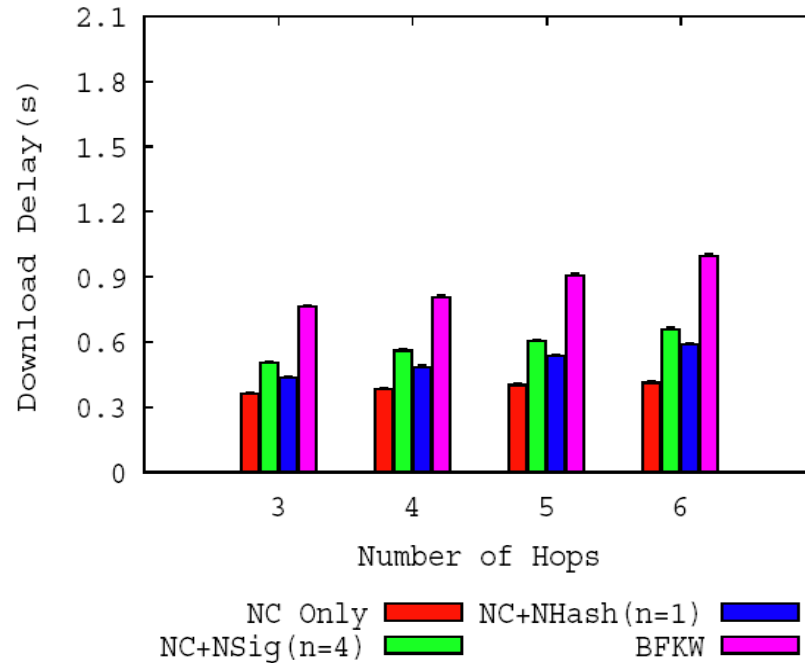
Simulation Setup

- Evaluate the performance in realistic network scenarios via simulation using the *experimental results*
- QualNet 3.9.5
 - Bandwidth: 2Mbps (broadcasting)
 - Data rate at source: 256Kbps
- Network Topology (static topology)
 - 1 Source/ 1 destination
 - Vary # of hops
- Reference Cases
 - NC_Only: Plain NC operations only
 - BFKW: Another homomorphic signature schemes applicable to NC
 - Boneh et al, PKC 2009 and Competitive to GKRR

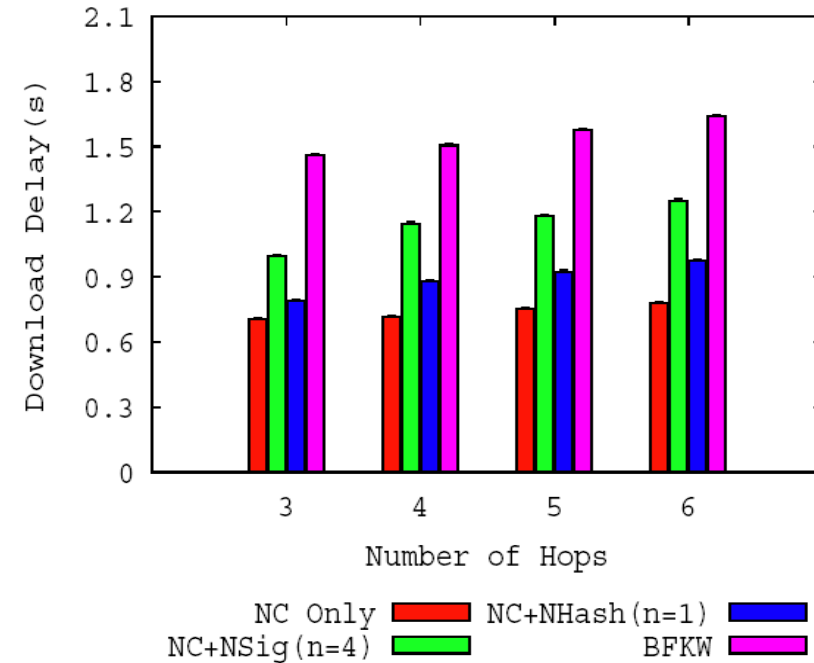


[BFKW]: D. Boneh, D. M. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In Public Key Cryptography, pages 68–87, 2009.

Simulation Results



(b) Download Delay (10KB, m=8)



(c) Download Delay (20KB, m=16)

- Delay increases with more hops between Src/ Dst
- NSig/NHash take less delay than BFKW

Related Works (1)

- Secure Random Checksums

- [J. Dong et al, 09]: TESLA-based approach
- [Gkantsidis et al, 06]: Secure Checksum

➔ **Requires** shared secret keys, network connectivity and synchronization.

Conclusion

- Studied feasibility of secure network coding schemes
 - Implemented the *theory* and measured processing overhead from experiments
 - Built a systematic framework based on implementation/experimental results
- Future works
 - More dynamic, realistic network scenarios, Systematic evaluations
 - Node mobility, various levels of attacks
 - Heterogeneous node types

Question?

Thank you