Padding for Orthogonality: Efficient Subspace Authentication for Network Coding

Peng Zhang, Yixin Jiang, Chuang Lin, Hongyi Yao,
Albert Wasef, Xuemin (Sherman) Shen

pzhang@csnet1.cs.tsinghua.edu.cn

April 12, 2011

Outline

Outline

Preliminary to Network Coding

Setting

The source needs to deliver m packets $\underline{\boldsymbol{x}}_1, \underline{\boldsymbol{x}}_2, \ldots, \underline{\boldsymbol{x}}_m$ to multiple receivers.

- When there are more than m packets to be sent, group these packets into multiple *generations*, each containing m packets.

$$\underbrace{\underline{\boldsymbol{x}}_1, \ldots, \underline{\boldsymbol{x}}_m}_{G_1}, \ldots, \underbrace{\underline{\boldsymbol{x}}_{(n-1)m+1}, \ldots, \underline{\boldsymbol{x}}_{nm}}_{G_n}, \cdots \tag{1}$$

- Denote each packet $\underline{\boldsymbol{x}}_i$ as a vector $(\underline{x}_{i,1}, \underline{x}_{i,2}, \ldots, \underline{x}_{i,n})$ over finite field $\mathbb{F}_q^n$

Preliminary to Network Coding

Source

- For each packet $\underline{\boldsymbol{x}}_i$, the source prefixes it with the i^{th} unit vector:

$$\boldsymbol{x}_i = (\overbrace{\underbrace{0, \cdots, 0}_{i-1}, 1, 0, \ldots, 0}^{m}, \underline{x}_{i,1}, \underline{x}_{i,2}, \cdots, \underline{x}_{i,n}) \tag{2}$$

- The source sends random linear combinations of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$:

$$\boldsymbol{y} = \sum_i^m \alpha_i \boldsymbol{x}_i = (\sum_i^m \alpha_i x_{i,1}, \sum_i^m \alpha_i x_{i,2}, \ldots, \sum_i^m \alpha_i x_{i,m+n}) \tag{3}$$

Preliminary to Network Coding

Forwarders

Forwards buffer its received packets $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_l$, and sends random linear combinations of them:

$$\boldsymbol{y} = \sum_i^l \beta_i \boldsymbol{y}_i = (\sum_i^l \beta_i y_{i,1}, \sum_i^l \beta_i y_{i,2}, \ldots, \sum_i^l \beta_i y_{i,m+n}) \tag{4}$$

Preliminary to Network Coding

Forwarders

Forwards buffer its received packets $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_l$, and sends random linear combinations of them:

$$\boldsymbol{y} = \sum_i^l \beta_i \boldsymbol{y}_i = (\sum_i^l \beta_i y_{i,1}, \sum_i^l \beta_i y_{i,2}, \ldots, \sum_i^l \beta_i y_{i,m+n}) \tag{4}$$

Receivers

After receiving m linearly independent packets $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m$, a receiver can decode as:

$$\begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \\ \vdots \\ \boldsymbol{z}_m \end{pmatrix} \xrightarrow[eliminations]{Guassian} \left(\begin{array}{c:c} & \underline{\boldsymbol{x}}_1 \\ I_{m\times m} & \underline{\boldsymbol{x}}_2 \\ & \vdots \\ & \underline{\boldsymbol{x}}_m \end{array}\right) \tag{5}$$

Benefits and Applications of Network Coding

Benefits

- Improved throughput in mutlicast transmissions
- Lower transmission delay
- Enhanced robustness to network dynamics

Benefits and Applications of Network Coding

Benefits

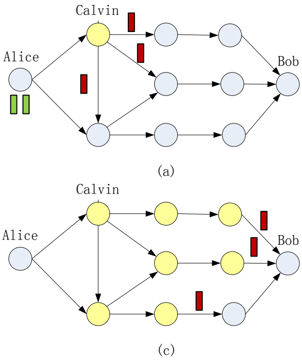
- Improved throughput in mutlicast transmissions
- Lower transmission delay
- Enhanced robustness to network dynamics

Applications

- P2P content distribution — Avalanche
- Wireless communications — COPE, MORE
- Distributed storage

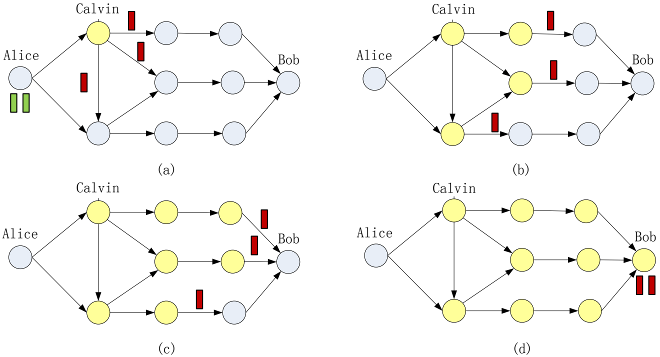
Pollution Attacks in Network Coding

Rapid dissemination of polluted packets in network coding:



Pollution Attacks in Network Coding

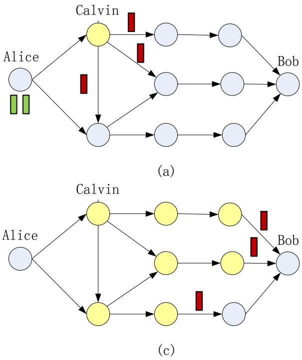
Rapid dissemination of polluted packets in network coding:



- A single polluted packet can corrupt bunches of good ones.

Pollution Attacks in Network Coding

Rapid dissemination of polluted packets in network coding:



- A single polluted packet can corrupt bunches of good ones.
- Traditional signature schemes cannot function with network coding.

Solutions

Cryptographic approaches

- Public-key-based: homomorphic hashing [Krohn04], homomorphic signature [Dan09]
- Symmetric-key-based: homomorphic MACs [Agrawa09], null keys [Kehdi09], ripple authentication [Li10]

Solutions

Cryptographic approaches

- Public-key-based: homomorphic hashing [Krohn04], homomorphic signature [Dan09]
- Symmetric-key-based: homomorphic MACs [Agrawa09], null keys [Kehdi09], ripple authentication [Li10]

Problems we observe

- Startup latency: in scenarios of multiple generations, the source should distribute secret keys, or signatures in prior to each generation.
- Security or efficiency:
  - Public-key-based — provably secure, but computationally expensive
  - Symmetric-key-based — efficient in computation but with a low security level or at the cost of sophisticated key distribution

Outline

Related Work — Subspace Authentication

Authenticate the subspace![Zhao06]

- Let $V = span(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m)$, then a packet $\boldsymbol{w}$ is valid *iff* it belongs to V.
- Let V' be the orthogonal space of V, the a vector $\boldsymbol{w}$ belongs to V *iff* $\boldsymbol{w} \perp V'$.

Related Work — Subspace Authentication

Authenticate the subspace![Zhao06]

- Let $V = span(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m)$, then a packet $\boldsymbol{w}$ is valid *iff* it belongs to V.
- Let V' be the orthogonal space of V, the a vector $\boldsymbol{w}$ belongs to V *iff* $\boldsymbol{w} \perp V'$.

$\Rightarrow$ to verify a packet $\boldsymbol{w}$, we can choose vector(s) $\boldsymbol{v}$ from V' and verify whether $\boldsymbol{w} \cdot \boldsymbol{v}^T = 0$.

Related Work — Subspace Authentication

Authenticate the subspace![Zhao06]

- Let $V = span(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m)$, then a packet $\boldsymbol{w}$ is valid *iff* it belongs to V.
- Let V' be the orthogonal space of V, the a vector $\boldsymbol{w}$ belongs to V *iff* $\boldsymbol{w} \perp V'$.

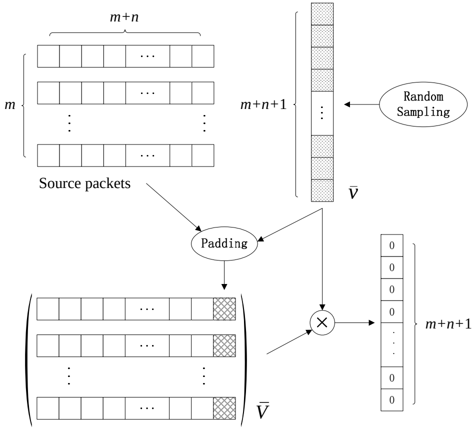
$\Rightarrow$ to verify a packet $\boldsymbol{w}$, we can choose vector(s) $\boldsymbol{v}$ from V' and verify whether $\boldsymbol{w} \cdot \boldsymbol{v}^T = 0$.

Startup Latency

For different generations, we should choose different $\boldsymbol{v}$'s and predistribute them to forwarders, just like most existing schemes.

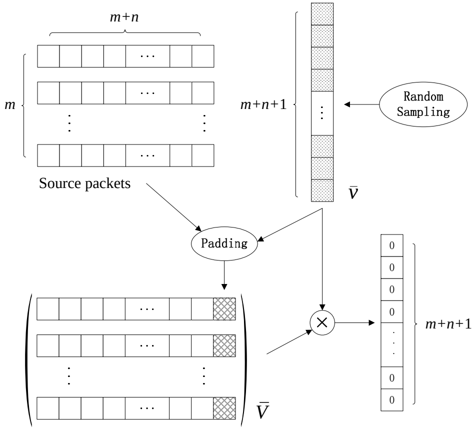
Basic Idea

Padding for orthogonality: Randomly choose a vector $\bar{v}$ of length $m+n+1$, and pad each packet with an extra symbol, so that its inner product with $\bar{v}$ equals zero.



Basic Idea

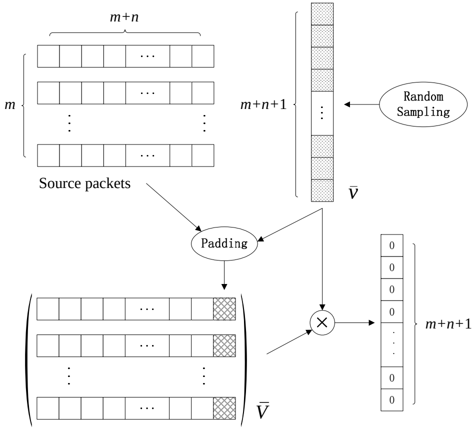
Padding for orthogonality: Randomly choose a vector $\bar{v}$ of length $m+n+1$, and pad each packet with an extra symbol, so that its inner product with $\bar{v}$ equals zero.



$\bar{v}$ doesn't depends on V.

Basic Idea

Padding for orthogonality: Randomly choose a vector $\bar{v}$ of length $m+n+1$, and pad each packet with an extra symbol, so that its inner product with $\bar{v}$ equals zero.



$\bar{v}$ doesn't depends on V.

No startup latency!

Homomorphic Subspace Signature (HSS)

Setup

- find a multiplicative cyclic group $\mathbb{G}$ of order q, and select a generator g for $\mathbb{G}$
- generate the private key $\boldsymbol{\beta} \xleftarrow{R} \mathbb{F}_q^{m+n}\mathbb{F}_q^*$
- generate the public key $\boldsymbol{h} = (g^{\beta_1}, \ldots, g^{\beta_{m+n+1}})$

Homomorphic Subspace Signature (HSS)

Setup

- find a multiplicative cyclic group $\mathbb{G}$ of order q, and select a generator g for $\mathbb{G}$
- generate the private key $\boldsymbol{\beta} \xleftarrow{R} \mathbb{F}_q^{m+n}\mathbb{F}_q^*$
- generate the public key $\boldsymbol{h} = (g^{\beta_1}, \ldots, g^{\beta_{m+n+1}})$

Sign

For each packet $\boldsymbol{x}$, calculate its signature as:

$$\sigma = -(\sum_{i=1}^{m+n} \beta_i x_i)/\beta_{m+n+1} \tag{6}$$

Let $\bar{\boldsymbol{x}} = (\boldsymbol{x}, \sigma)$ denote the signed packet, then we have $\bar{\boldsymbol{x}} \cdot \boldsymbol{\beta}^T = 0$.

Homomorphic Subspace Signature (HSS)

Combine

To combine signed packets $\bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_l$ using coefficients $\alpha_1, \ldots, \alpha_l$, simply calculate $\bar{\boldsymbol{x}} = \sum_{i=1}^{l} \alpha_i \bar{\boldsymbol{x}}_i$.

Homomorphic Subspace Signature (HSS)

Combine

To combine signed packets $\bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_l$ using coefficients $\alpha_1, \ldots, \alpha_l$, simply calculate $\bar{\boldsymbol{x}} = \sum_{i=1}^{l} \alpha_i \bar{\boldsymbol{x}}_i$.

Verify

To verify a signed packet $\bar{\boldsymbol{x}}$, calculate

$$\delta = \boldsymbol{h}^{\bar{\boldsymbol{x}}} \triangleq \prod_{i=1}^{m+n+1} h_i^{\bar{x}_i} \tag{7}$$

Accept $\bar{\boldsymbol{x}}$ if $\delta = 1$, or reject it otherwise.

Homomorphic Subspace Signature (HSS)

Combine

To combine signed packets $\bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_l$ using coefficients $\alpha_1, \ldots, \alpha_l$, simply calculate $\bar{\boldsymbol{x}} = \sum_{i=1}^{l} \alpha_i \bar{\boldsymbol{x}}_i$.

Verify

To verify a signed packet $\bar{\boldsymbol{x}}$, calculate

$$\delta = \boldsymbol{h}^{\bar{\boldsymbol{x}}} \triangleq \prod_{i=1}^{m+n+1} h_i^{\bar{x}_i} \tag{7}$$

Accept $\bar{\boldsymbol{x}}$ if $\delta = 1$, or reject it otherwise.

- breaking HSS is at least as hard as solving discrete logarithm problems
- Modular exponentiation over finite field is expensive.

Homomorphic Subspace MAC (HSM)

Setup

- Generate r secret keys $\boldsymbol{\gamma_1}, \ldots, \boldsymbol{\gamma_r}$:
  $\boldsymbol{\gamma}_i = (\gamma_{i,1}, \ldots, \gamma_{i,m+n+1}) \xleftarrow{R} \mathbb{F}_q^{m+n}\mathbb{F}_q^*$, $i = 1, \ldots, r$
- For each secret key, assign it to every node with some probability

Homomorphic Subspace MAC (HSM)

Setup

- Generate r secret keys $\boldsymbol{\gamma_1}, \ldots, \boldsymbol{\gamma_r}$:
  $\boldsymbol{\gamma}_i = (\gamma_{i,1}, \ldots, \gamma_{i,m+n+1}) \xleftarrow{R} \mathbb{F}_q^{m+n}\mathbb{F}_q^*$, $i = 1, \ldots, r$
- For each secret key, assign it to every node with some probability

MAC

For each packet $\boldsymbol{x}$, calculate r MACs $t_1, \ldots, t_r$:

$$t_i = -(\sum_{j=1}^{m+n} \gamma_{i,j} x_j)/\gamma_{i,m+n+1}, i = 1, \ldots, r \tag{8}$$

Let $\bar{\boldsymbol{x}} = (\boldsymbol{x}, t_1, \ldots, t_r)$ denote the MAC-carrying packet.

Homomorphic Subspace MAC (HSM)

Combine

To combine signed packets $\bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_l$ using coefficients $\alpha_1, \ldots, \alpha_l$, simply calculate $\bar{\boldsymbol{x}} = \sum_{i=1}^{l} \alpha_i \bar{\boldsymbol{x}}_i$.

Homomorphic Subspace MAC (HSM)

Combine

To combine signed packets $\bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_l$ using coefficients $\alpha_1, \ldots, \alpha_l$, simply calculate $\bar{\boldsymbol{x}} = \sum_{i=1}^{l} \alpha_i \bar{\boldsymbol{x}}_i$.

Verify

For tagged packet $\bar{\boldsymbol{x}}$ and calculate

$$\xi_i = \sum_{j=1}^{m+n} \gamma_{i,j} \bar{x}_j + \gamma_{i,m+n+1} \bar{x}_{m+n+i} \qquad (9)$$

for each secret key γ_i the node has.
Accept $\bar{\boldsymbol{x}}$ if all $\xi_i = 0$, or reject it otherwise.

Homomorphic Subspace MAC (HSM)

Combine

To combine signed packets $\bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_l$ using coefficients $\alpha_1, \ldots, \alpha_l$, simply calculate $\bar{\boldsymbol{x}} = \sum_{i=1}^{l} \alpha_i \bar{\boldsymbol{x}}_i$.

Verify

For tagged packet $\bar{\boldsymbol{x}}$ and calculate

$$\xi_i = \sum_{j=1}^{m+n} \gamma_{i,j}\bar{x}_j + \gamma_{i,m+n+1}\bar{x}_{m+n+i} \tag{9}$$

for each secret key γ_i the node has.
Accept $\bar{\boldsymbol{x}}$ if all $\xi_i = 0$, or reject it otherwise.

HSM is efficient in computation.

Security of HSM — Colluding Adversaries

Security

To resist a collusion of c adversaries, the number of secret keys (MACs per packet) should be no less than $e(c+1)^2 \ln n$, where n is the number of nodes in the network.

Security of HSM — Colluding Adversaries

Security

To resist a collusion of c adversaries, the number of secret keys (MACs per packet) should be no less than $e(c+1)^2 \ln n$, where n is the number of nodes in the network.

The number of MACs per packet doesn't scale with the network size!

Security of HSM — Colluding Adversaries

Double-Random Key Distribution

- Intuition: the number of secret keys and the number of MACs per packet needn't to be the same.
- Approach: the source randomly selects $l < r$ secret keys for each generation, and calculates l MACs using these keys for each packet.

Security of HSM — Colluding Adversaries

Double-Random Key Distribution

- Intuition: the number of secret keys and the number of MACs per packet needn't to be the same.
- Approach: the source randomly selects $l < r$ secret keys for each generation, and calculates l MACs using these keys for each packet.

To resist a collusion of c adversaries with probability of $1 - \epsilon$, we only need to attach $l = \frac{1}{1-\delta} e(c+1) \ln \frac{1}{\epsilon}$ MACs for each packet. Note that l has no relationship with the network size n.

Security of HSM — Tag Pollution

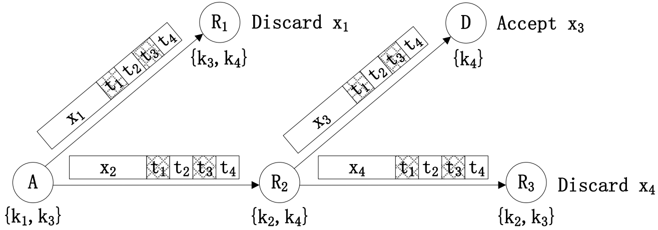
HSM is vulnerable to tag pollution.

- The adversary modifies the tags (MACs) carried by packets rather than the contents of them.
- It is possible that a packet with polluted tags travels multiple hops until it is finally detected and discarded, which can result in a waste of network bandwidth.

Security of HSM — Tag Pollution

HSM is vulnerable to tag pollution.

- The adversary modifies the tags (MACs) carried by packets rather than the contents of them.
- It is possible that a packet with polluted tags travels multiple hops until it is finally detected and discarded, which can result in a waste of network bandwidth.



Brief Summary of Baseline Schemes

HSS

- provably secure based on the hardness assumption of discrete logarithm problem
- rather computationally expensive due to the calculations of modular exponentiations

Brief Summary of Baseline Schemes

HSS

- provably secure based on the hardness assumption of discrete logarithm problem
- rather computationally expensive due to the calculations of modular exponentiations

HSM

- more efficient in computation.
- vulnerable to colluding adversaries and tag pollution.

Brief Summary of Baseline Schemes

HSS

- provably secure based on the hardness assumption of discrete logarithm problem
- rather computationally expensive due to the calculations of modular exponentiations

HSM

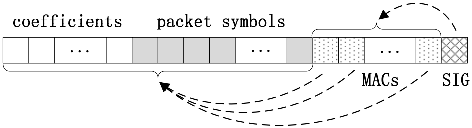
- more efficient in computation.
- vulnerable to colluding adversaries and tag pollution.

Can we combine the advantages of HSS and HSM, and strike a better balance between security and efficiency?

Outline

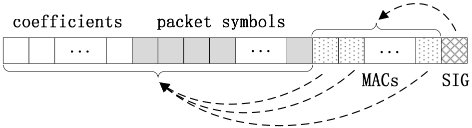
The Unified MacSig Approach

The packet content is authenticated by homomorphic subspace MACs, and these MACs are further authenticated by a homomorphic subspace signature.



The Unified MacSig Approach

The packet content is authenticated by homomorphic subspace MACs, and these MACs are further authenticated by a homomorphic subspace signature.

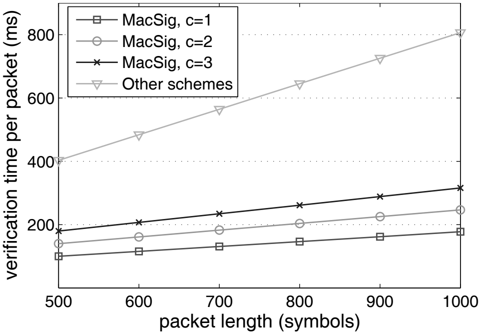


Benefits of MacSig

- It can resists normal pollution attacks, as well as tag pollution
- It is more efficient than pure signature schemes
- It incur a moderate bandwidth overhead

Computation Overhead

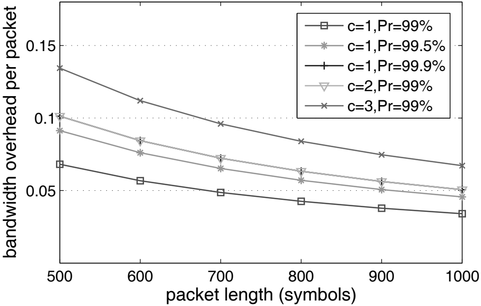
To verify a packet in MacSig, $m + l + 1$ exponentiations and $(m + n + 1)l$ multiplications are needed.



The verification process of MacSig is 2 to 4 times faster than those of the other three.

Bandwidth Overhead

The per-packet bandwidth overhead per packet includes l MACs, l MAC key indexes, and a signature.



When the packet size is larger than 700 symbols and the number of colluding adversaries is less than 3, the per-packet bandwidth overhead sits between 5% and 10%.

Outline

Conclusion

Conclusion

- We present a novel idea termed *"padding for orthogonality"* for network coding authentication

Conclusion

- We present a novel idea termed *"padding for orthogonality"* for network coding authentication
- We design a public-key based signature scheme and a symmetric-key based MAC scheme, which can both effectively contain pollution attacks

Conclusion

- We present a novel idea termed *"padding for orthogonality"* for network coding authentication
- We design a public-key based signature scheme and a symmetric-key based MAC scheme, which can both effectively contain pollution attacks
- We combine them to propose a unified scheme termed *MacSig*, which can thwart both normal pollution and tag pollution attacks in an efficient way.

Q&A

Thank You!