

# Digital Signatures for Flows and Multicasts

by Chung Kei Wong and Simon S. Lam  
in *IEEE/ACM Transactions on Networking*, August  
1999

# Digital Signature

- Examples: RSA, DSA
- Provide authenticity, integrity and non-repudiation
- How to sign/verify?
  - signing key  $k_s$ , verification key  $k_v$ , message digest  $h(m)$
  - $signature = sig(h(m), k_s)$
  - $verify(signature, h(m), k_v) = True/False$
- Signing & verification operations are slow compared to symmetric key operations

# Motivation

- Traditional network applications (circa 1998)
  - message-oriented unicast, e.g., email, file transfer, client-server
- Emerging network applications
  - flow-oriented, e.g., audio, video, stock quotes
  - multicast, e.g., teleconference, software distribution
- Problem: How to sign efficiently?
  - high-speed transmissions
  - real-time generated flows
  - delay-sensitive packet flows

# All-or-nothing flows

- ❑ The signer generates a message digest of the entire flow (file) and signs the message digest
  
- ❑ But most Internet applications do not create *all-or-nothing* flows
  - a flow is sent as a sequence of packets
  - each packet is used as soon as it is received

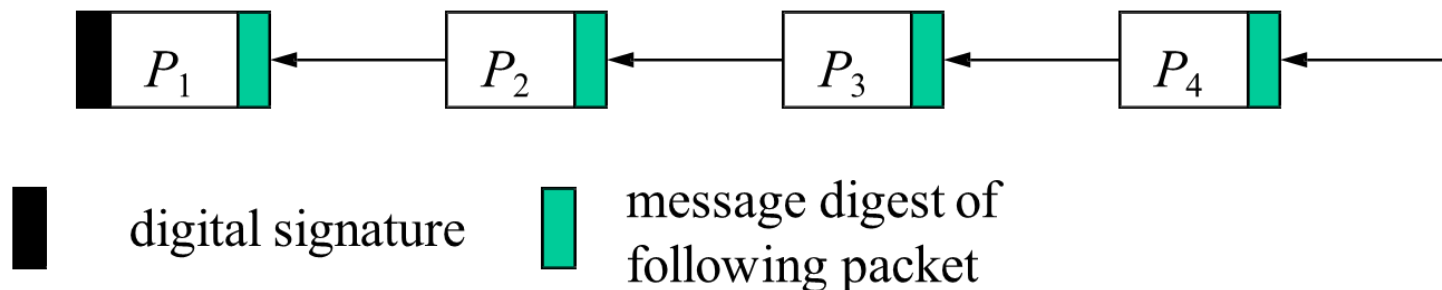
# Sign-each Approach

- ❑ A flow is a sequence of data packets
- ❑ Sign each packet individually
- ❑ Inefficient: *one signing/verification operation per packet*
- ❑ Rates on a Pentium-II 300 MHz using 100% processing time (with 512-bit modulus)

Packet size (bytes)	Rate (packets/sec)			
	Signing		Verification	
	RSA	DSA	RSA	DSA
512	78.8	176	2180	128
1024	78.7	175	1960	127

# Prior work on signing digital streams

- [Gennaro and Rohatgi 1997]
- One signing/verification op for an entire flow—only the first packet is signed
  - Each packet contains authentication info for next
- Verification of each packet depends on previous ones
  - *Reliable delivery* required

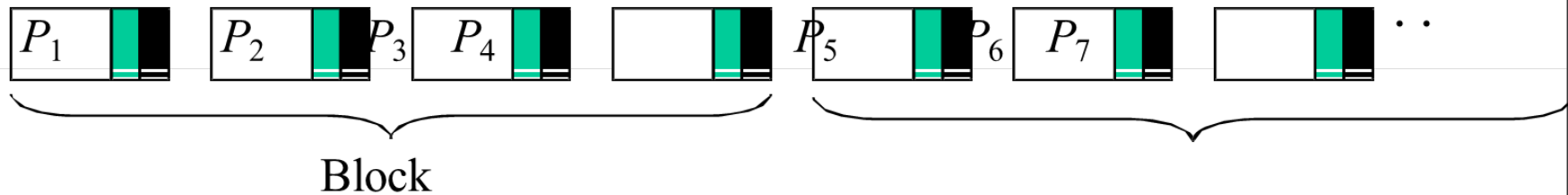


# Flow Signing Problem

- ❑ Each packet may be used as soon as it is received
- ❑ Subsequences of a flow are received and used
  - best-effort delivery, e.g., UDP, IP multicast
  - different needs/capabilities, e.g., layered video
- ❑ How to efficiently sign flows with each packet being *individually verifiable*?

# Our Approach: Chaining

- Partition a flow into blocks of packets
  - Sign the digest of each block instead of each packet individually
- Each packet carries its own authentication information to prove it is in the block
  - Authentication info provided by chaining



Block signature



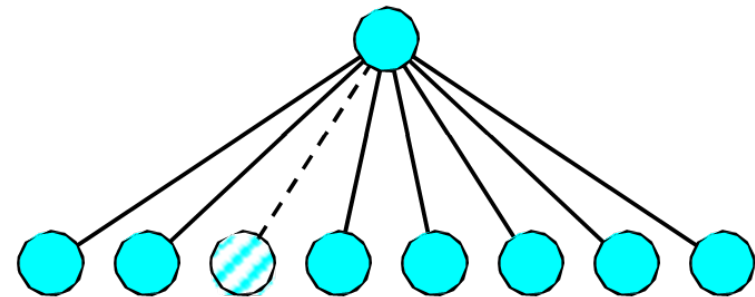
Chaining info





# Star Chaining - Signing

Block digest  $D_{1-8} = h(D_1, \dots, D_8)$



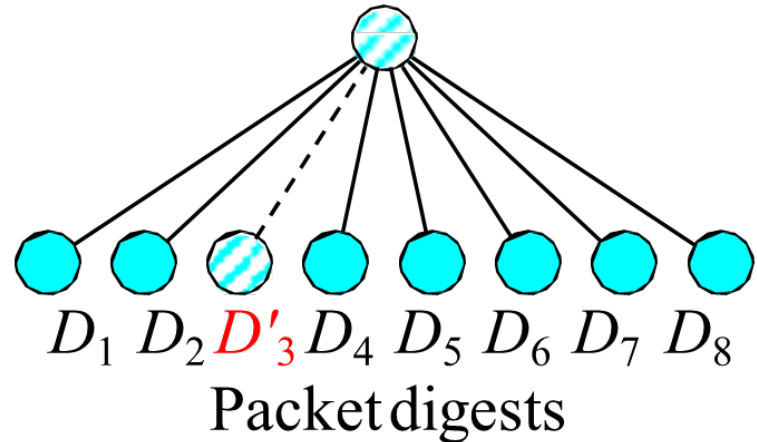
Packet digests  $D_1 D_2 D_3 D_4 D_5 D_6 D_7 D_8$

- ❑ Block signature =  $sign(D_{1-8})$
- ❑ Packet signature for packet  $P_3$ :  $sign(D_{1-8}), D_1, D_2, D_4, \dots, D_8$
- ❑ Chaining overhead is  $O(\text{block size})$

# Star Chaining - Verification

- Verifying first received packet (say  $P_3$ )

Block digest  $D'_{1-8} = h(D_1, D_2, D'_3, D_4, \dots, D_8)$



- $verify(D'_{1-8}, sign(D_{1-8}))$

- Caching of verified nodes

- no verification op for other packets in the block

# Tree Chaining - Signing

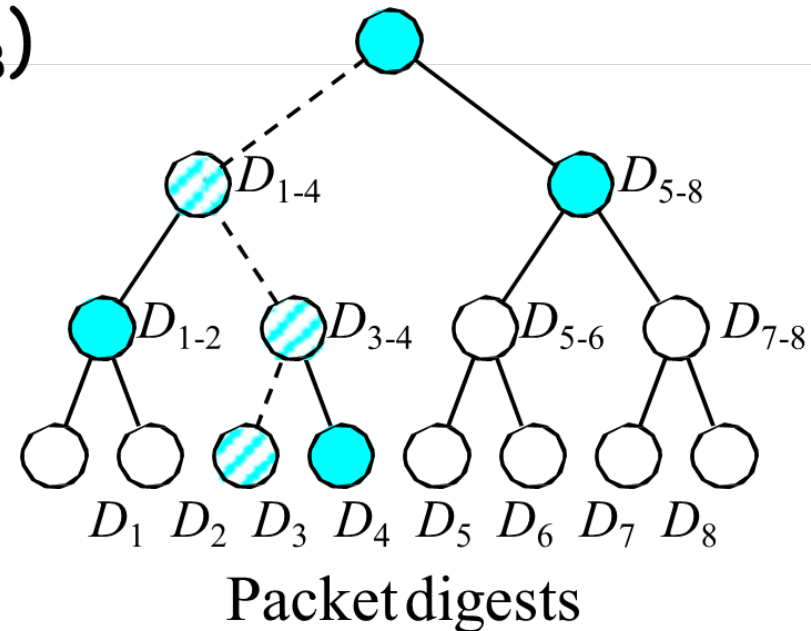
□ [Merkle 1989]

□ Block signature =  $sign(D_{1-8})$

□ Packet signature for  
packet  $P_3$ :  
 $sign(D_{1-8}), D_4, D_{1-2}, D_{5-8}$

□ Chaining overhead is  
 $O(\log(\text{block size}))$

Block digest  $D_{1-8} = h(D_{1-4}, D_{5-8})$

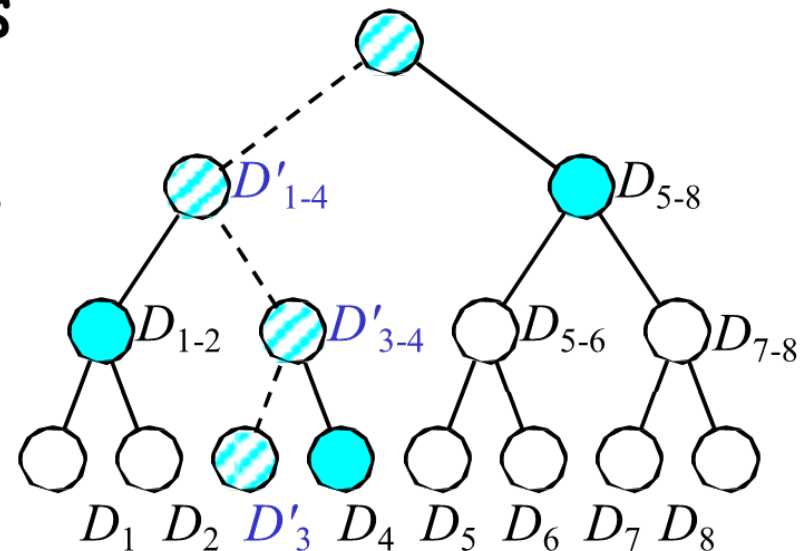


# Tree Chaining - Verification

- Verifying first received packet (say  $P_3$ )
  - $verify(D'_{1-8}, sign(D_{1-8}))$

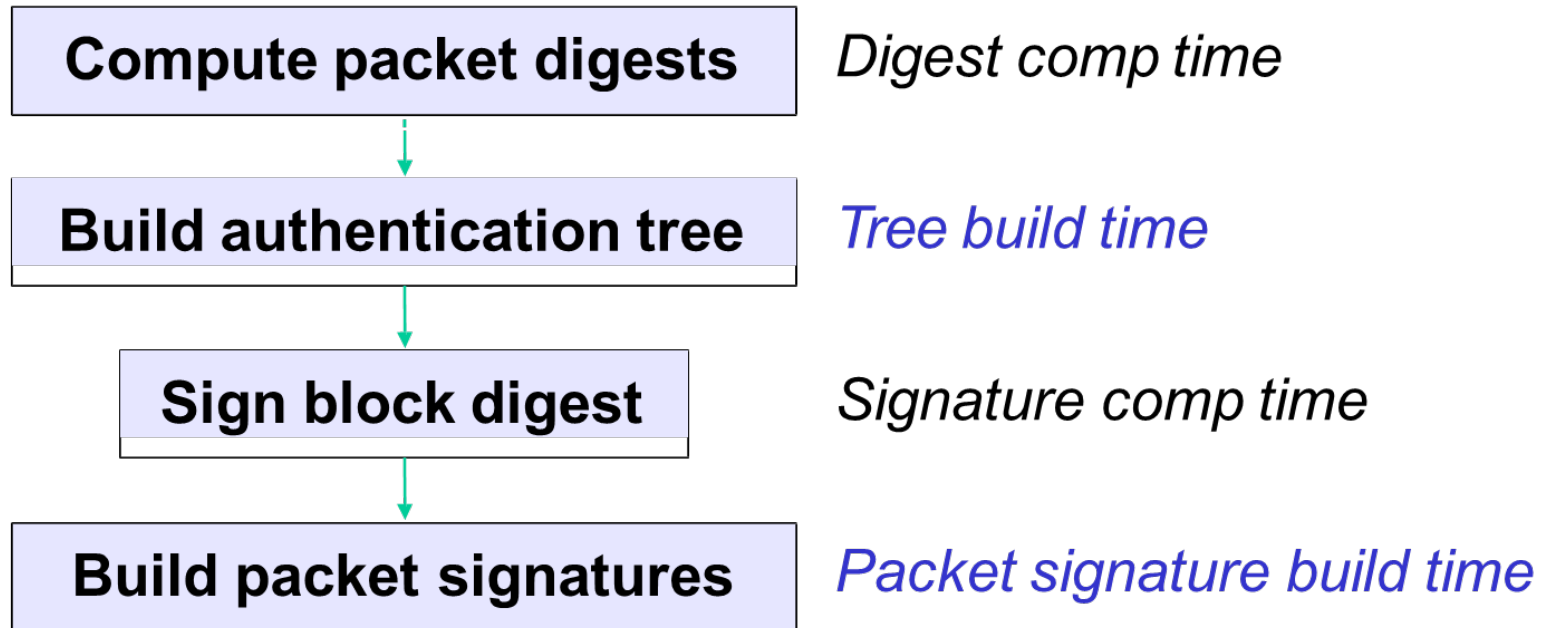
Block digest  $D'_{1-8} = h(D'_{1-4}, D_{5-8})$

- Caching of verified nodes
  - no verification op for other packets in the block



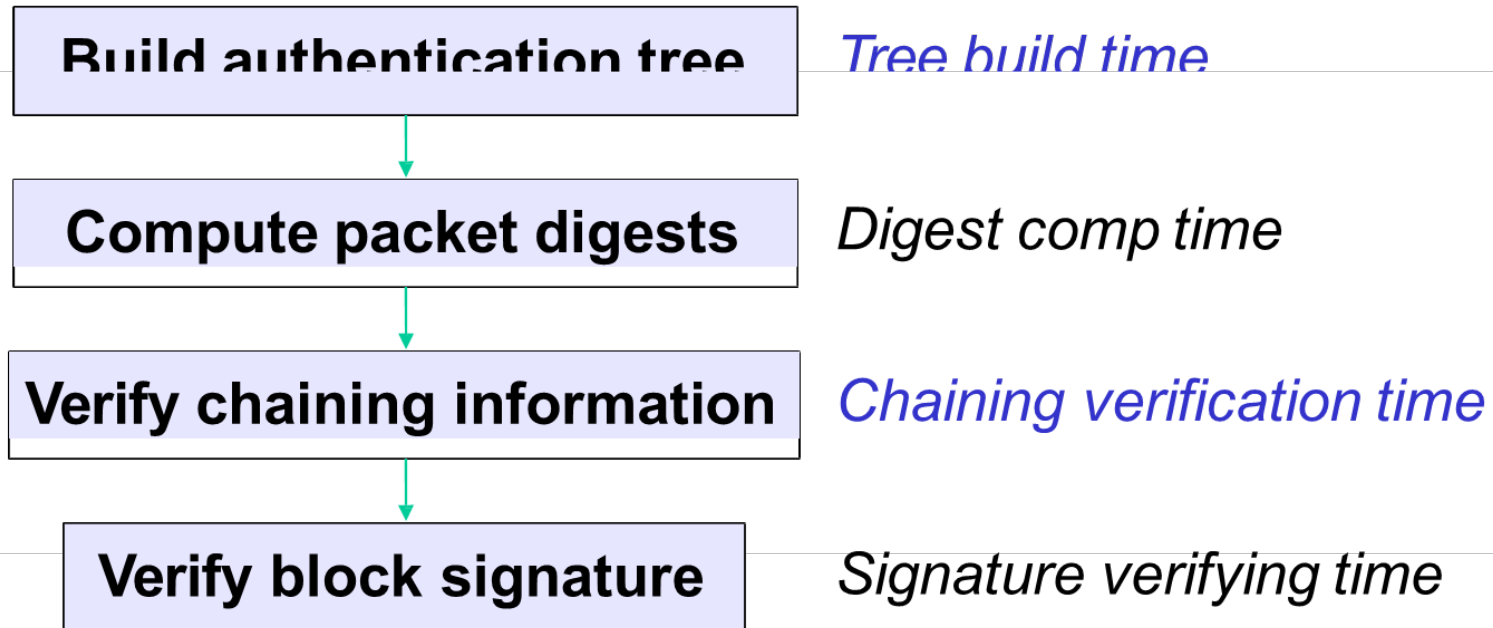
Packet digests

# Chaining Technique: Signer Overhead



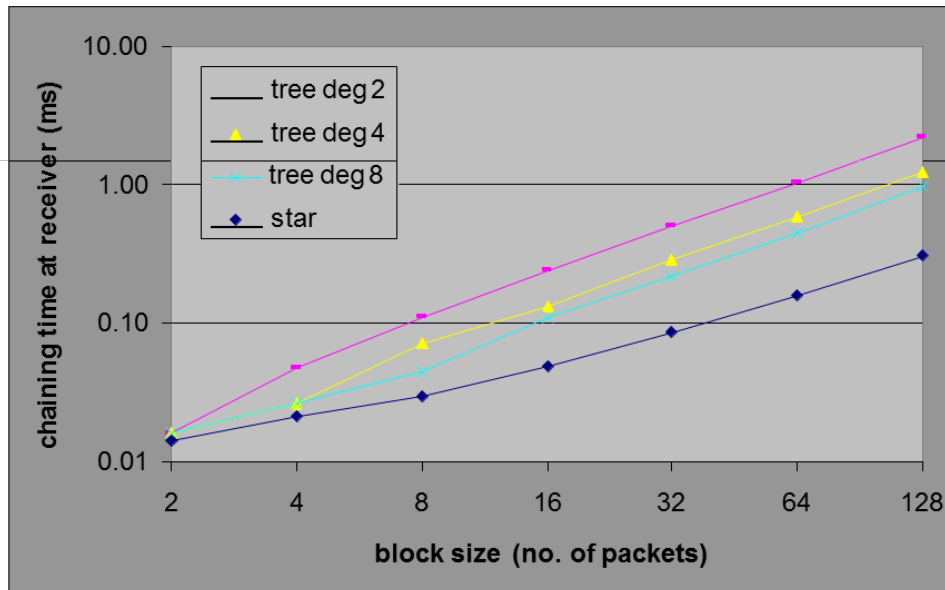
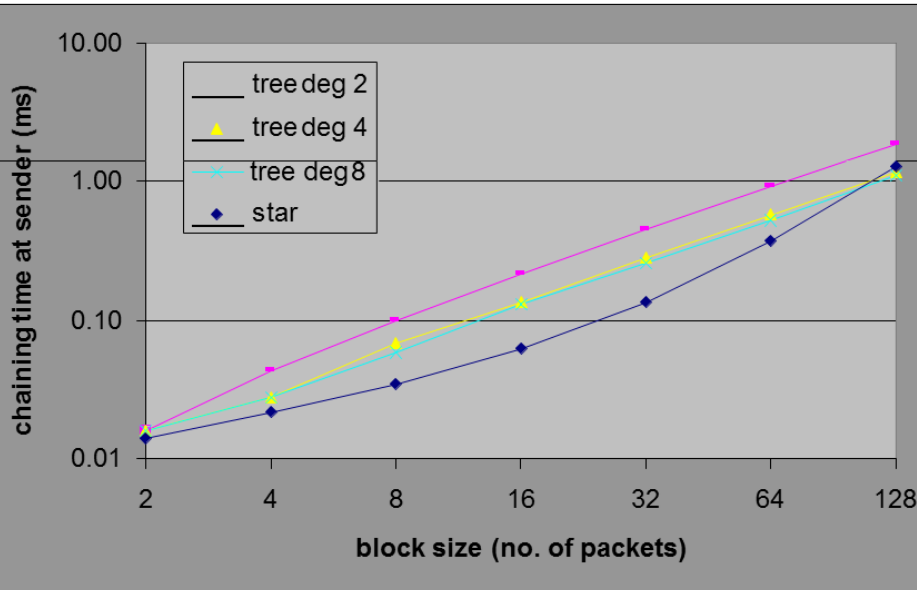
*Chaining time = Tree build time + Packet signature build time*

# Chaining Technique: Verifier Overhead



***Chaining time = Tree build time + Chaining verification time***

# Chaining Time Overheads

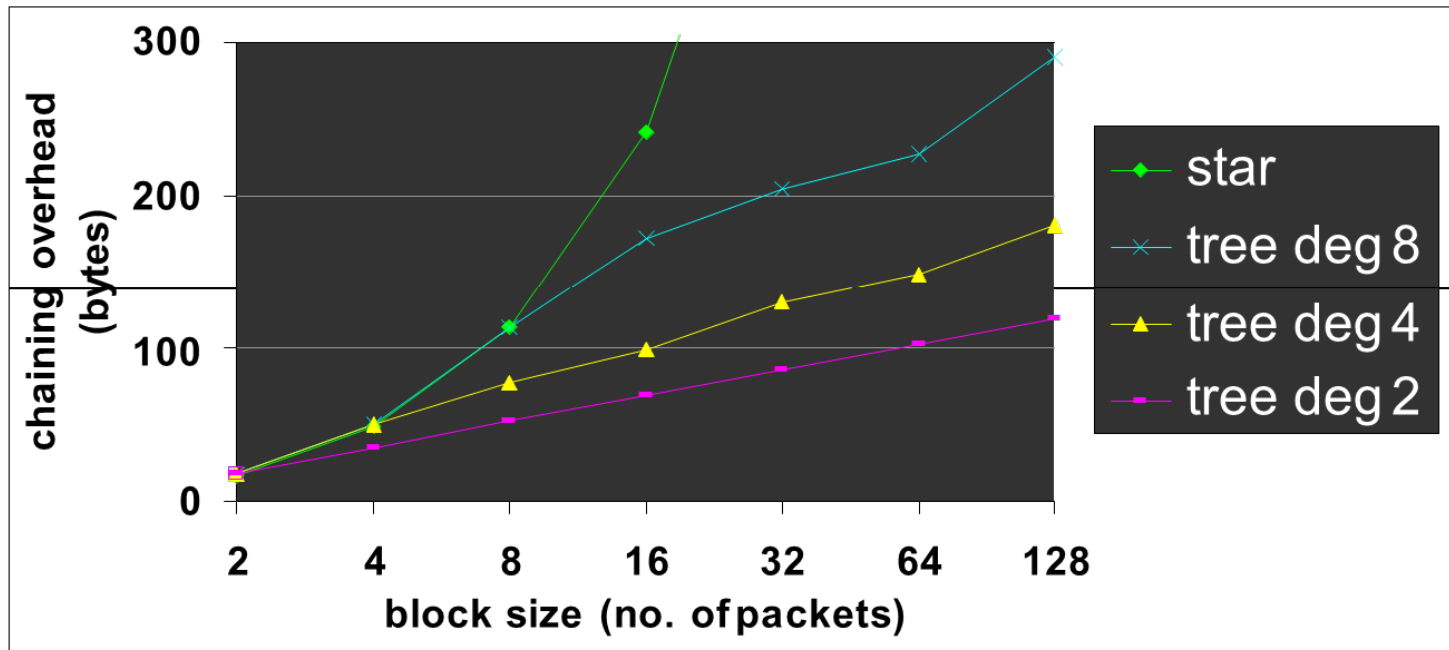


at sender

at receiver

- ❑ Overheads increase linearly with block size (in log scale)
- ❑ Much smaller than signing/verification times

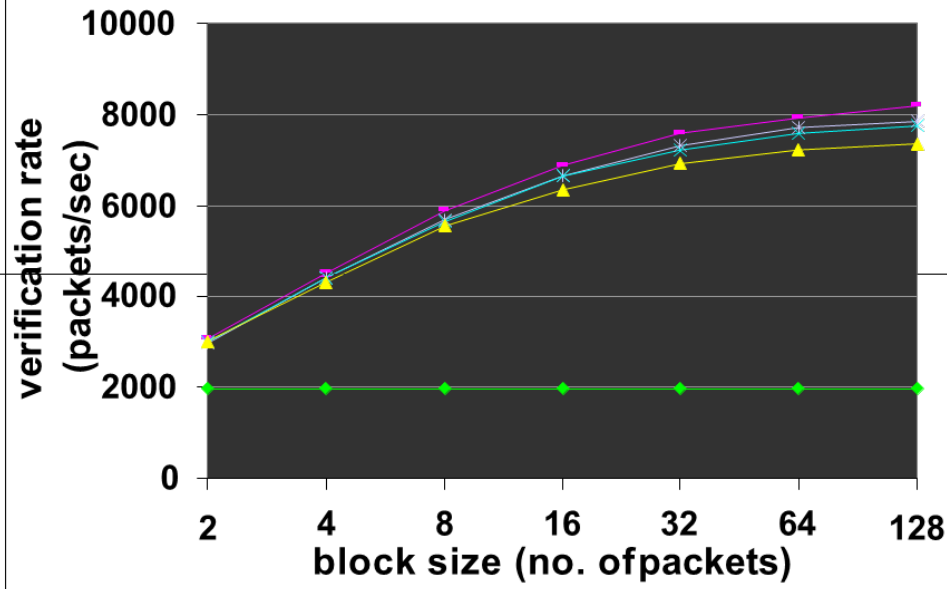
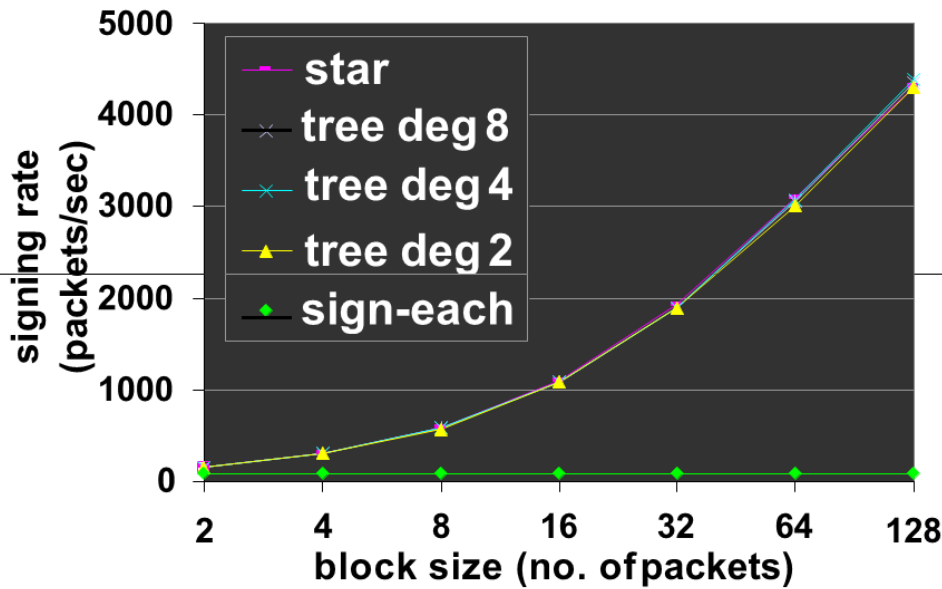
# Chaining Overhead Size



- ❑ Smallest when tree degree is 2
- ❑ Increases linearly with logarithm of block size
- ❑ Packet signature = block signature + chaining overhead

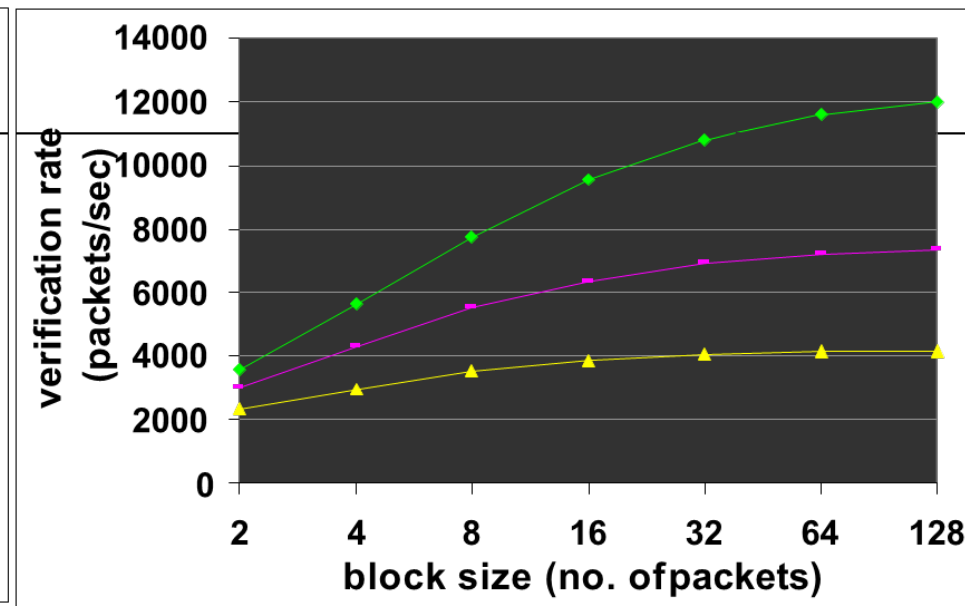
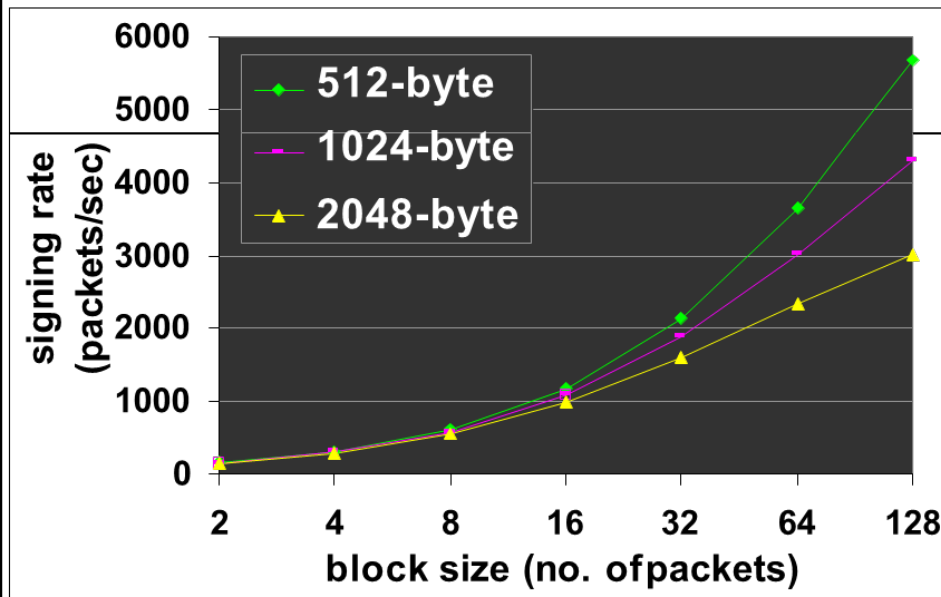


# Flow Signing/Verification Rates



- ❑ 1024-byte packets, **RSA** with 512-bit modulus
- ❑ Increases with block size
- ❑ Varies only slightly with tree degree
  - we recommend degree 2 tree chaining

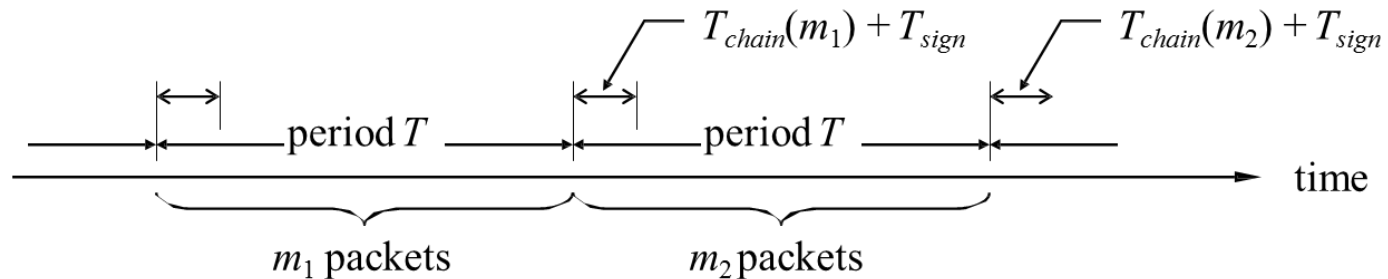
# Flow Signing/Verification Rates



- Degree two tree, RSA with 512-bit modulus, three different packet sizes

# Real-time Generated Flows

- Fixed block size for non-real-time generated flows
- Fixed time period  $T$  for real-time generated flows
- Bounded delay signing since for any packet  
 $delay \leq T + T_{chain} + T_{sign}$



- $T$  should be larger than  $T_{chain} + T_{sign}$
- $delay$  cannot be smaller than  $2(T_{chain} + T_{sign})$

# Selecting a Signature Scheme

- RSA: signing rate not high enough
  
- DSA: both rates not high and verification rate < signing rate
  - In a group, receivers may have widely different resources, e.g., PDAs, notebooks, desktops
  
- We proposed several extensions to FFS  
[Feige, Fiat and Shamir 1986]

# FFS Signer

- choose two large primes  $p$  and  $q$
- compute modulus  $n = pq$
- choose integers  $v_1, \dots, v_k$   
 $s_1, \dots, s_k$

such that  $s_i^2 = v_i^{-1} \pmod n$

- signing key is  $\{s_1, \dots, s_k, n\}$
- verification key is  $\{v_1, \dots, v_k, n\}$

# How to Sign Message $m$

□ choose  $t$  random integers,  $r_1, \dots, r_t$ , between 1 and  $n$

□ compute  $x_i = r_i^2 \bmod n$ , for  $i = 1, \dots, t$

□ compute message digest  $h(m, x_1, \dots, x_t)$

where function  $h(\cdot)$  is public knowledge and produces a digest of at least  $k \times t$  bits

let  $\{b_{ij}\}$  be the first  $k \times t$  bits of the digest

□ compute  $y_i = r_i \times (s_1^{b_{i1}} \times \dots \times s_k^{b_{ik}}) \bmod n$

for  $i = 1, \dots, t$

□ signature of  $m$  consists of

$\{y_i\}$  and  $\{b_{ij}\}$  for  $i = 1, \dots, t$  and  $j = 1, \dots, k$

# How to Verify Signature of Message $m$

□ signature of  $m$

$\{y_i\}$  and  $\{b_{ij}\}$  for  $i = 1, \dots, t$  and  $j = 1, \dots, k$

□ compute  $z_i = y_i^2 \times (v_1^{b_{i1}} \times \dots \times v_k^{b_{ik}}) \bmod n$

for  $i = 1, \dots, t$

it can be shown that  $z_i$  is equal to  $x_i$  at the signer

□ signature is **valid** if and only if the first

$x$  bits of  $(x_1, \dots, x_t)$  are equal to the  $\{y_{ij}\}$  received in signature

# FFS(k,t)

- security level increases with
  - size of modulus (or size of primes and )
  - value of product  $kt$
- key size is  $(k+1) \times |n|$   
assuming  $|n| = |v_i|$  or  $|s_i|$  in bits
- signature size is  $t \times |n| + k \times t$  bits  
minimized for  $t=1$



# FFS key and signature sizes

FFS SIGNING/VERIFICATION KEY AND SIGNATURE SIZES (BYTES) WITH 512-BIT MODULUS

	$t = 1$		$t = 2$		$t = 4$	
	key	sig	key	sig	key	sig
$kt = 64$	4160	72	2112	136	1088	264
$kt = 128$	8256	80	4160	144	2112	272

For a fixed  $kt$  product, signature size is minimized for  $t=1$ , but key size is maximized

# eFFS Signature Scheme

- Several extensions to FFS [Feige, Fiat and Shamir 1986]
  - Faster signing
    - Chinese remainder theorem (*crt*)
    - Precomputation (*4-bit, 8-bit*)
  - Faster verification
    - Small verification key (*sv-key*) [Micali & Shamir 1990]
  - Adjustable and incremental verification
    - multilevel signature
    - lower security level with less processor time at receiver
    - security level can be increased later by more processor time

# eFFS extension (1)

## □ Chinese remainder theorem

instead of  $y_i = r_i x (s_1^{b_{i1}} x \dots x s_k^{b_{ik}}) \bmod n$

signer computes

$$a_i = r_i x (s_1^{b_{i1}} x \dots x s_k^{b_{ik}}) \bmod p$$

$$b_i = r_i x (s_1^{b_{i1}} x \dots x s_k^{b_{ik}}) \bmod q$$

$$y_i = ((a_i - b_i) x q x q_p^{-1} + b_i) \bmod n$$

where  $q_p^{-1}$  denotes  $q^{-1} \bmod p$ ,

- multiplications in mod  $p$  and mod  $q$  faster than in mod  $n$

## □ Only signer knows $p$ and $q$

## eFFS extension (2)

- small verification key [Micali & Shamir]:

use first  $k$  prime numbers that satisfy

$$s^2 = p^{-1} \pmod{n}$$

where  $p$  is prime and  $s$  is an integer

- faster verifying time and smaller key size

## eFFS extension (3)

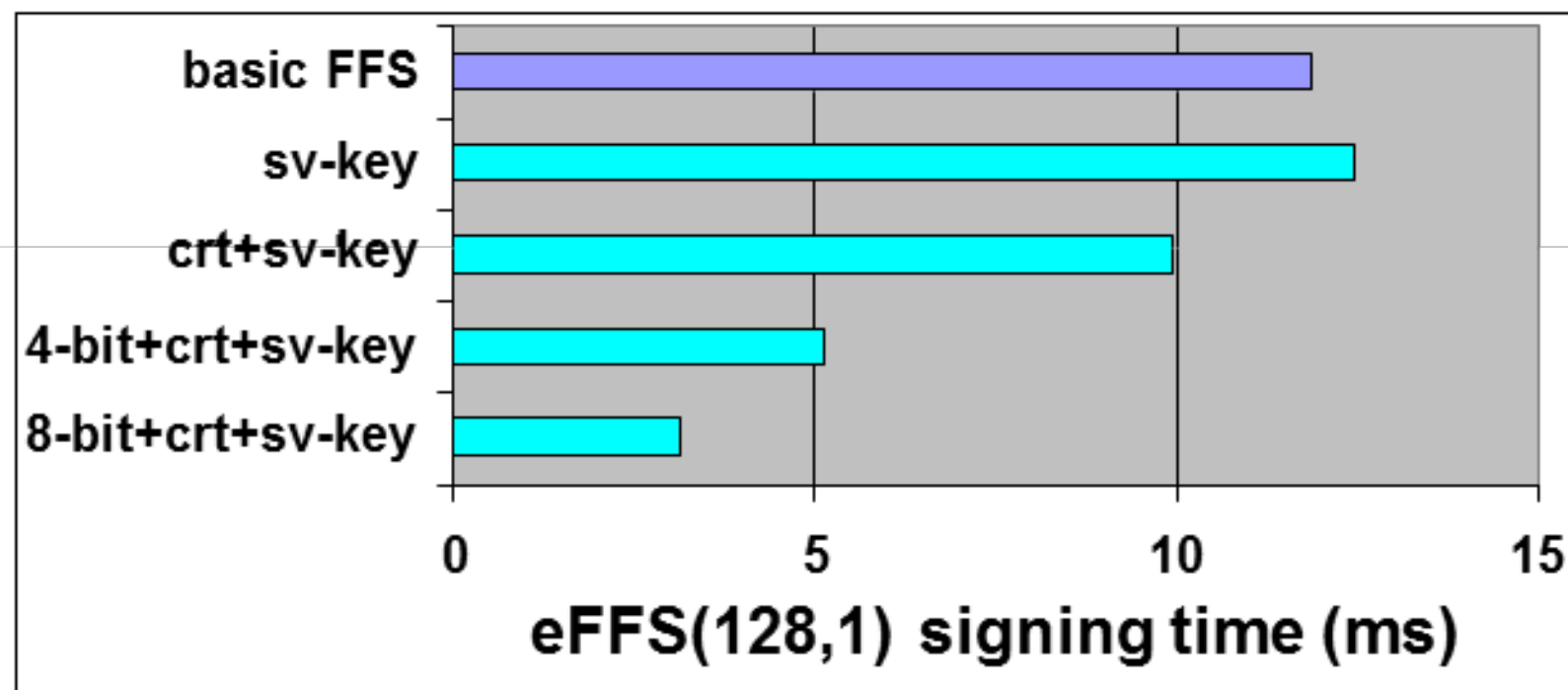
□ To compute  $y_i = r_i \times (s_1^{b_{i1}} \times \dots \times s_k^{b_{ik}}) \bmod n$   
for  $i = 1, \dots, t$

□ precomputation of  $(s_1^{b_{i1}} \times \dots \times s_k^{b_{ik}})$

additional memory of 31 KB and 261 KB  
required for 4-bit and 8-bit precomp  
respectively

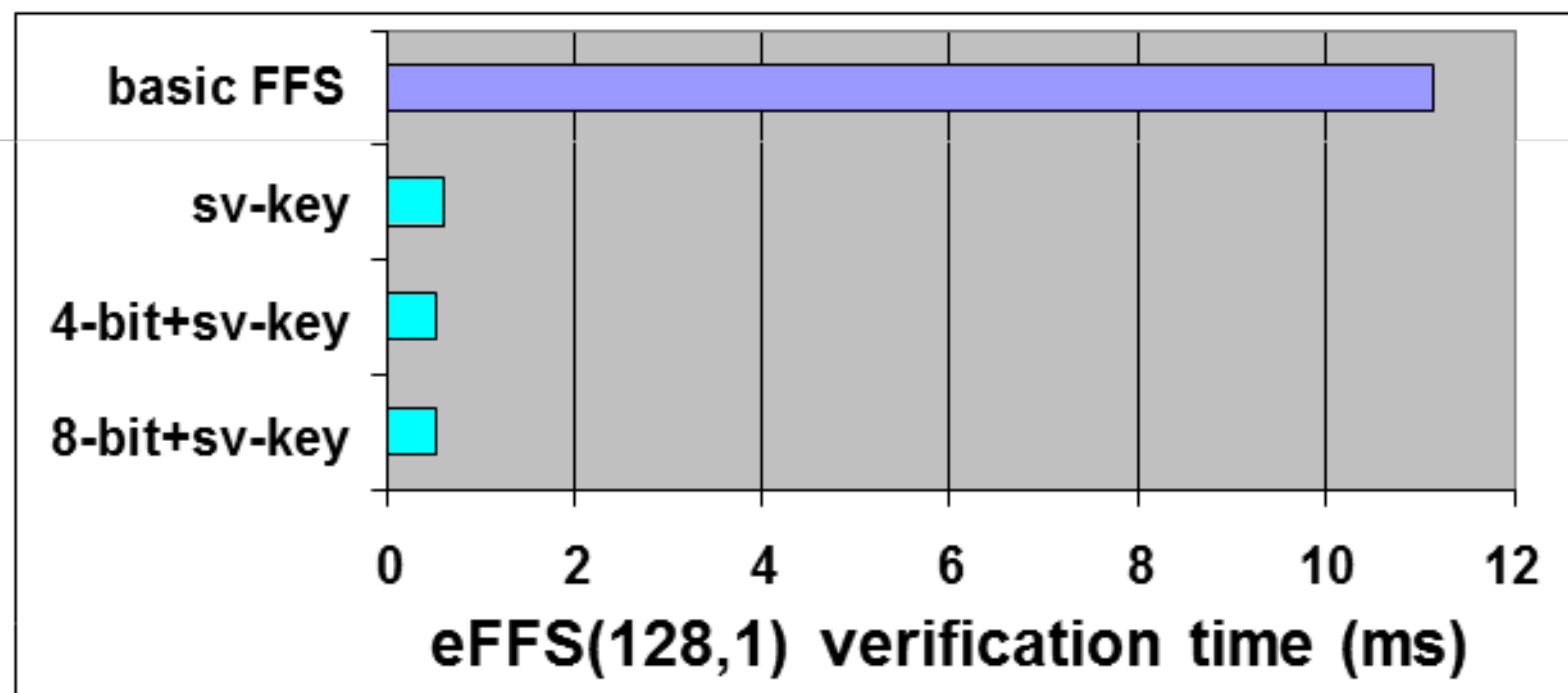
only minor improvement at verifier when  
used with small v-key

# eFFS - Signing



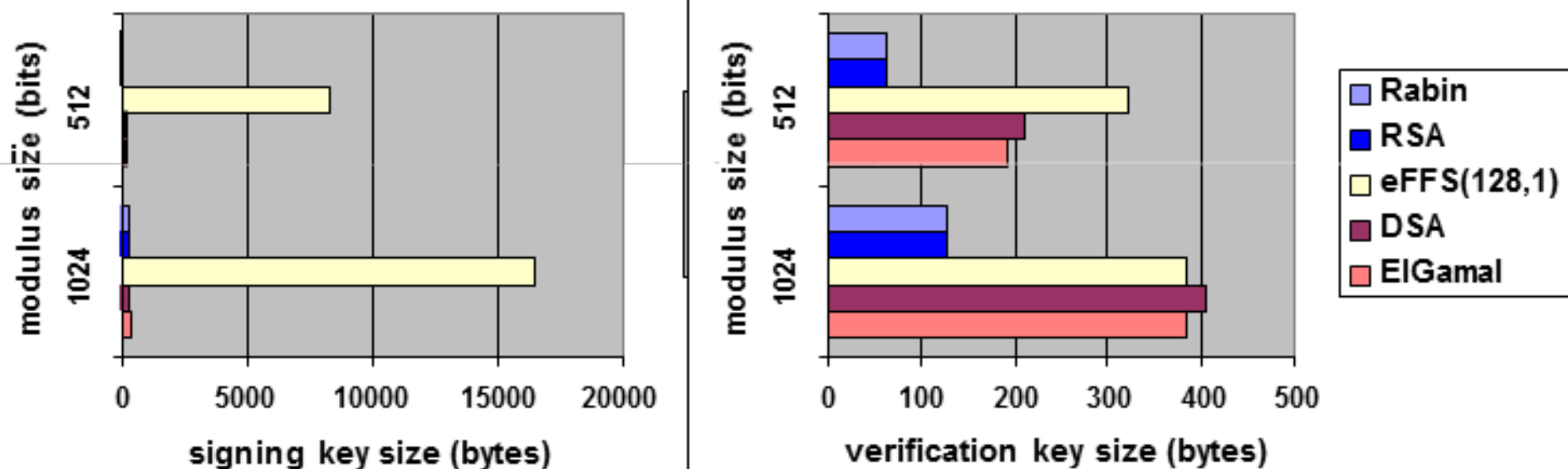
- ❑ *sv-key* does not reduce signing time
- ❑ *crt* reduces signing time by 10-20%
- ❑ *8-bit + crt* reduces signing time by 60-70%

# eFFS - Verification



- **sv-key** reduces verification time by 90%
- **4-bit** or **8-bit** slightly reduces verification time

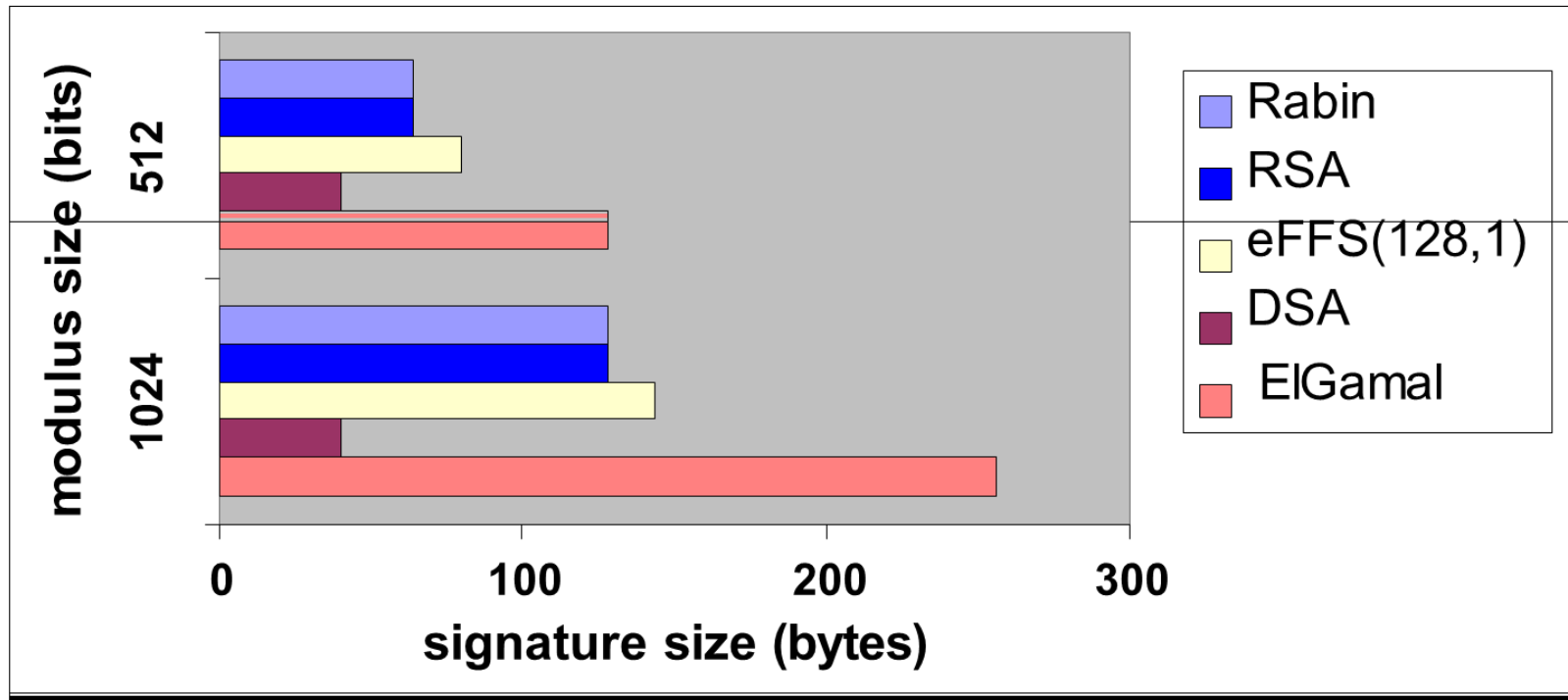
# eFFS Key Size



- ❑ Large signing key 8000-17000 bytes
  - private to signer
- ❑ Verification key 300-400 bytes

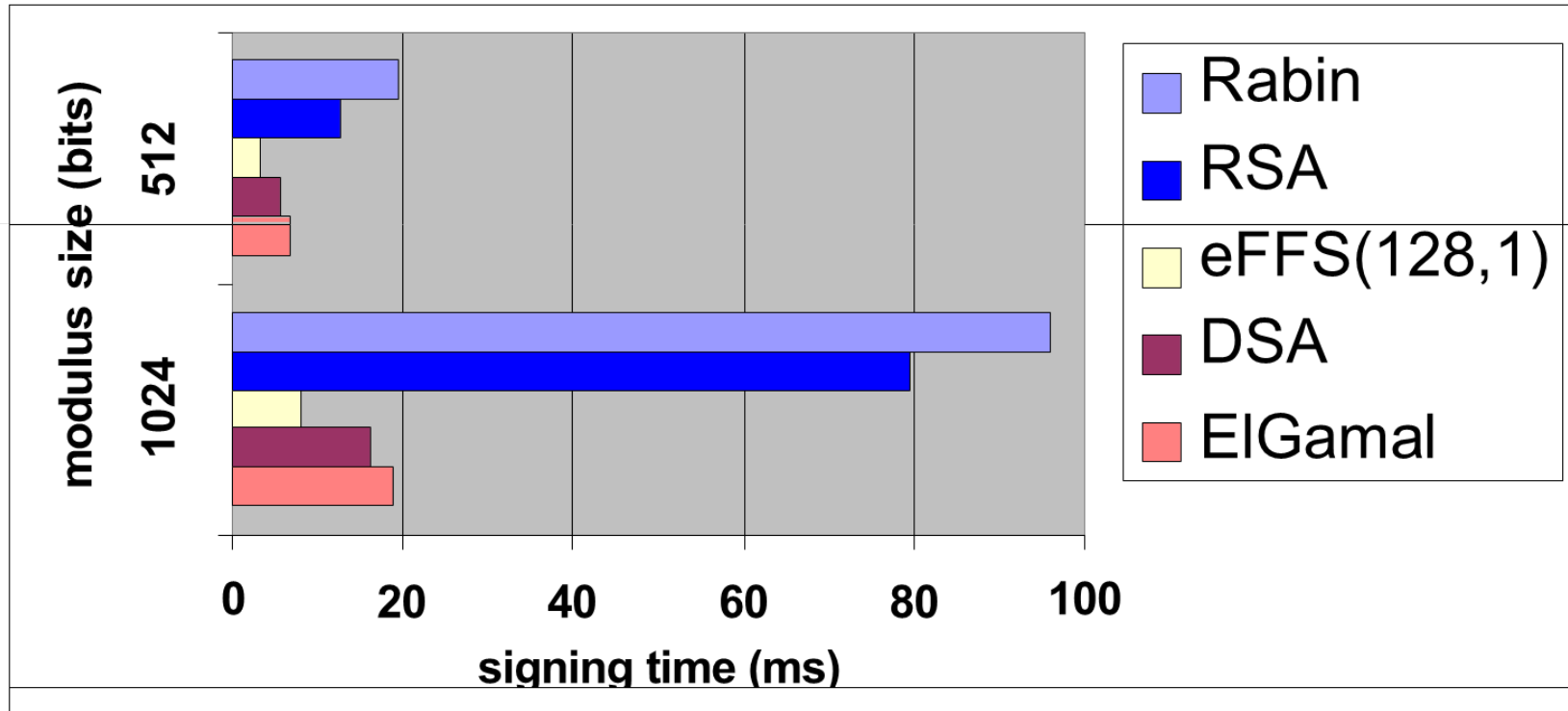


# eFFS Signature Size



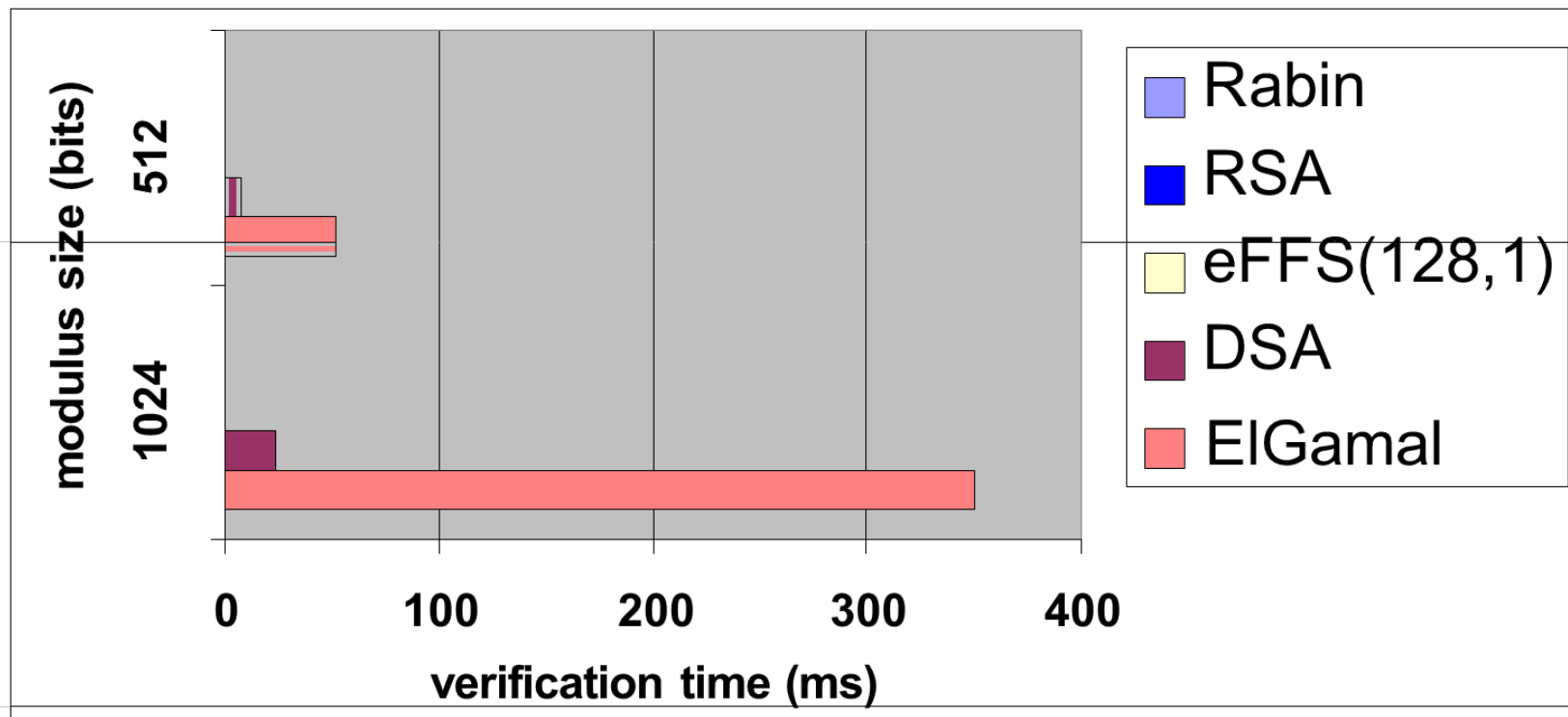
- Signature size comparable to RSA and Rabin

# Signing Time Comparison



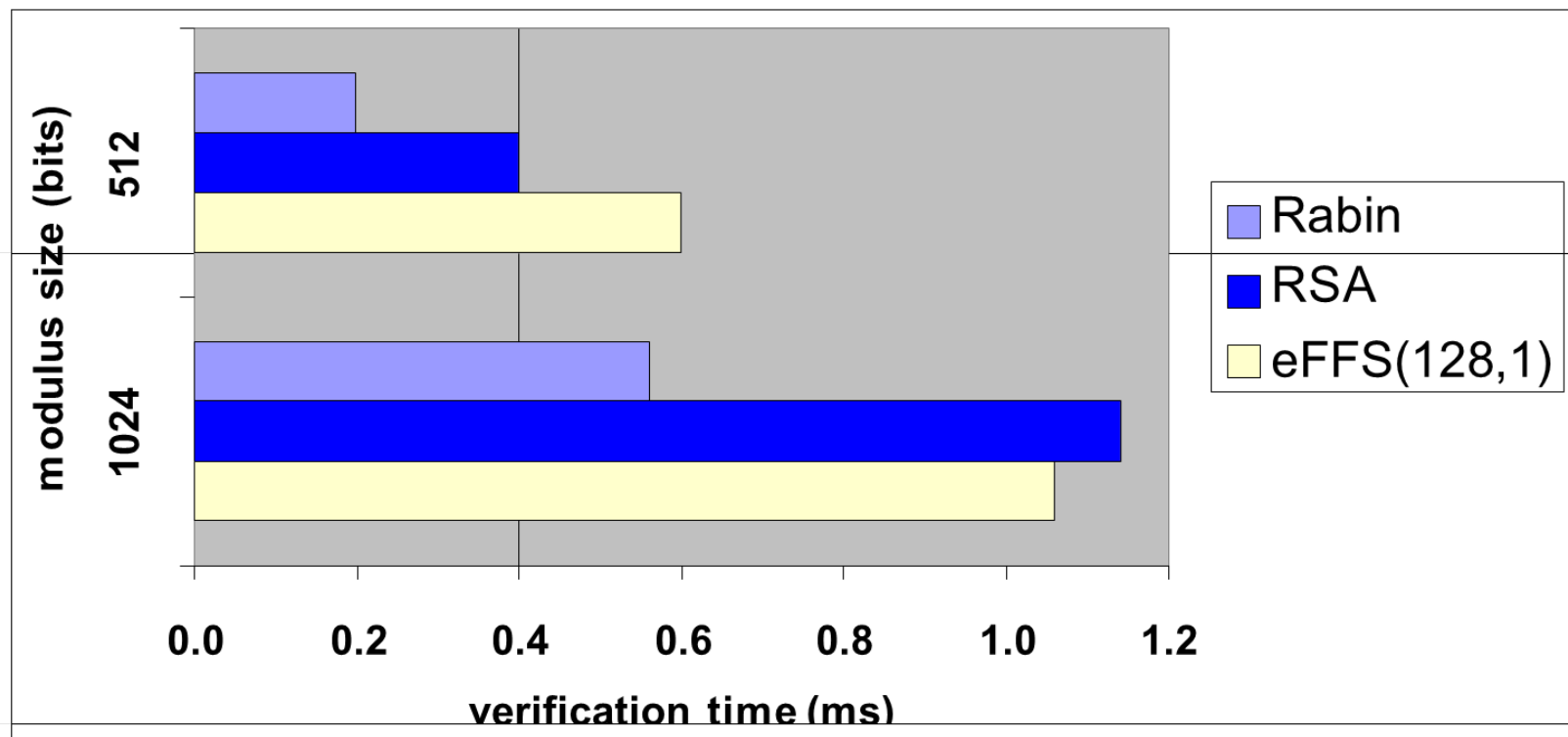
- ❑ 8-bit + crt + sv-key extensions
- ❑ eFFS has the *smallest signing time*

# Verification Time Comparison



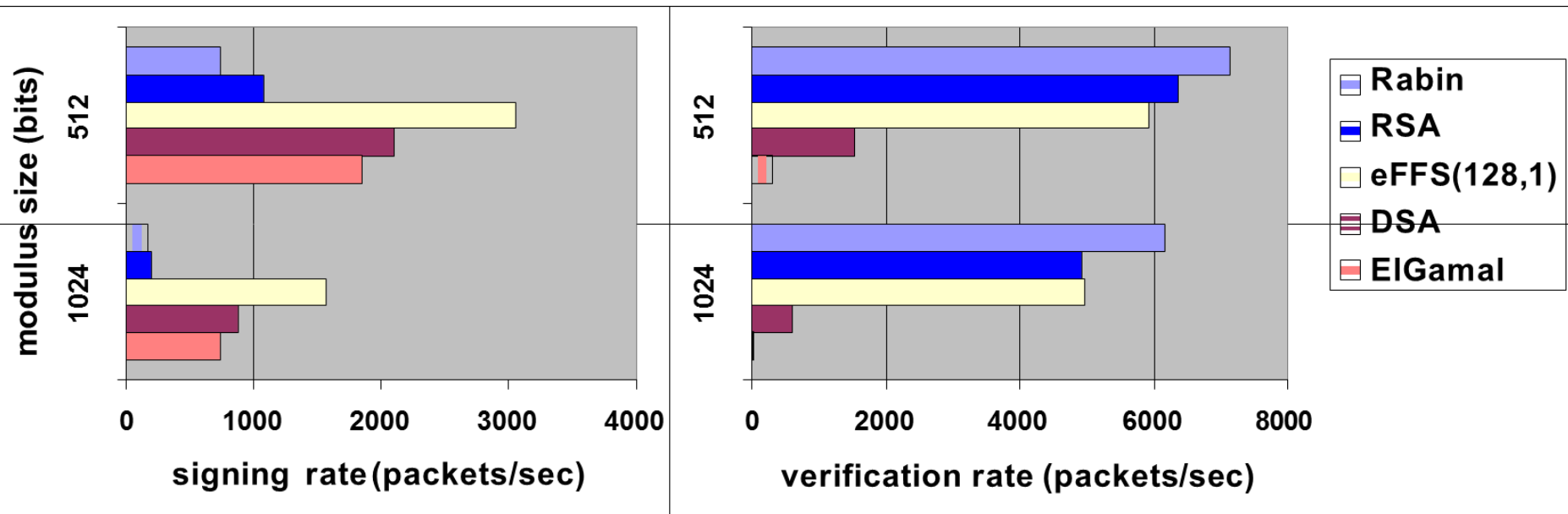
- ❑ DSA and ElGamal verification times very large
- ❑ Rabin, RSA and eFFS too small to see

# Verification Time Comparison



- eFFS verification time comparable to RSA (Rabin most efficient verification)

# Flow Signing/Verification Rates



- 1024-byte packets, block size 16, degree two tree chaining
- eFFS has highest signing rate
- eFFS verification rate comparable to RSA

# eFFS Adjustable and Incremental Verification

- Security level of  $eFFS(k,t)$  depends on modulus size and product  $k$ 
  - same  $kt$  and modulus size  $\sim$  same security level
  
- Adjustable and incremental verification
  - using  $t > 1$  with additional info in signature
  - up to  $t$  steps
  - adjustable and incremental:  
receiver verifies steps one by one

# eFFS Adjustable and Incremental Verification (cont.)

- $t$ -level signature includes  $\{x_i\}$  for  $i = 2, \dots, t$   
note that  $\{x_i\}$  can be computed from original signature together with verification key
- verify a  $t$ -level signature **at security level  $l \leq t$** ,
  - (1) **compute**  $z_i = y_i \times (v_1^{b_{i1}} \times \dots \times v_k^{b_{ik}}) \bmod n$  for  $i = 1, \dots, l$ ,
  - (2) **verify** that the first  $k \times t$  bits of  $h(m, z_1, x_2, \dots, x_t)$  are equal to the  $\{b_{ij}\}$  received, and  $z_2, \dots, z_l$  are equal to  $x_2, \dots, x_l$

# eFFS Adjustable and Incremental Verification (cont.)

- increase security level from  $l_1$  to  $l_2$ ,
  - (1) compute  $z_i = y_i^2 x (v_1^{b_{i1}} \times \dots \times v_k^{b_{ik}}) \bmod n$  for  $i = l_1 + 1, \dots, l_2$ ,
  - (2) verify that  $z_{l_1+1}, \dots, z_{l_2}$  are equal to  $x_{l_1+1}, \dots, x_{l_2}$



# Incremental signing times

eFFS  $t$ -LEVEL SIGNATURE SIGNING TIMES (MILLISECONDS)

	$kt$ product		
	$kt = 32$	$kt = 64$	$kt = 128$
1-level signature	1.47	2.02	3.14
2-level signature		2.87	3.98
4-level signature			5.67

2-level signature takes less time to sign than two 1-level signatures

# Incremental verification times

eFFS INCREMENTAL VERIFICATION TIMES (MILLISECONDS) FOR  $kt = 128$ .

(a) 2-LEVEL SIGNATURE. (b) 4-LEVEL SIGNATURE.

To	level 1	level 2
From level 0	0.42	0.81
From level 1		0.40

(a)

To	level 1	level 2	level 3	level 4
From level 0	0.34	0.63	0.93	1.22
From level 1		0.30	0.60	0.89
From level 2			0.30	0.60
From level 3				0.31

(b)

# Conclusions

- Flow signing/verification procedures
  - much more efficient than sign-each
  - small communication overhead
  - can be used by a sender that signs a large number of packets to different receivers
    - there is no requirement that the packets belong to a flow but if they do, verification is also more efficient
  
- eFFS digital signature scheme
  - most efficient signing compared to RSA, Rabin, DSA, and ElGamal
  - highly efficient verification and comparable to RSA (only Rabin is more efficient)
  - adjustable and incremental verification

End