High Performance BGP Security: Algorithms and Architectures

Mehmet Adalier, Kotikalapudi Sriram,

Oliver Borchert, Kyehwan Lee, Doug Montgomery

Email: madalier@antarateknik.com; ksriram@nist.gov

Acknowledgements: Randy Bush, IIJ

NANOG-69, February 2017, Washington DC

Antara Teknik LLC contribution to this work was supported by the National Institute of Standards and Technology (NIST) under an SBIR cooperative agreement 70NANB14H289. NIST's research was supported by the Department of Homeland Security under the Secure Protocols for the Routing Infrastructure (SPRI) program and the NIST Information Technology Laboratory Internet Infrastructure Protection Program.

BGP Vulnerabilities

Border Gateway Protocol is vulnerable to malicious attacks that target the control plane

- Prefix/sub-prefix hijacks
  - Steers traffic away from legitimate servers
- Prefix squatting
  - Hijacks a not-in-service prefix and sets up spam servers
- AS path modification (Man-in-the Middle) attacks
  - Modifies AS path causing data to flow via the attacker
- Route leaks
  - Announces routes in violation of ISP policy, thereby redirecting traffic via the attacker

The exploitations commonly result in DoS, spam, misrouting of data traffic, eavesdropping on user data, etc.

Objectives of the IETF SIDR WG Security Solution

- Verify that the originating AS shown in the announcement is authorized to originate the prefix [RFC 6811]
- Verify that the BGP announcement did indeed traverse a sequence of ASs as shown by AS path in the announcement -- BGPsec [1]
- Provide protection from withdrawal suppression and malicious replay attacks [4]
- Accommodate special cases such as:
  - Transparent Route Servers (RS) at Internet Exchange Points (IXP)
  - An AS confederation must be visible externally as a single public AS
- Provide route-leak detection capability (as best as possible) [5]

IETF/IESG recently approved “BGPsec Protocol Specification” draft as Proposed Standard.

https://tools.ietf.org/html/draft-ietf-sidr-bgpsec-protocol-22

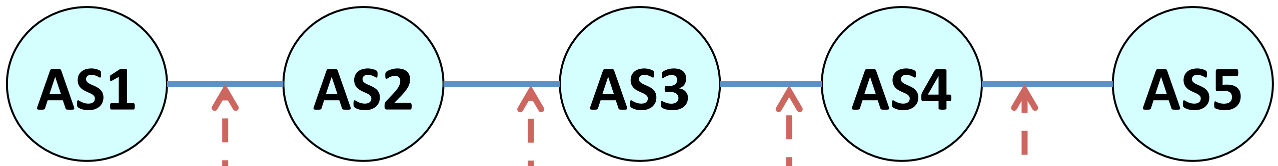
Key Elements of the Security Solution

- Hierarchical certificate chain for resource allocations
  - IP address blocks (prefixes)
  - Autonomous system numbers
- Resource PKI (RPKI) repository
- Prefix owner signs a Route Origin Authorization (ROA) authorizing an AS to originate one or more prefixes
  - ROA: {ASN; Prefix1, maxLength1; Prefix2, maxLength2}
- Each BGPsec speaker uses its private key to sign updates it is propagating to its peer AS
- Receiving router validates the origin using ROA information
- Receiving router also performs path validation by verifying the AS path signatures in updates

RPKI provides provenance and integrity.

AS Path Protection:
Basic Principle of BGPsec AS Path Signing

p = 10.1.0.0/16



BGPsec Update:

p, (**T=AS2***) AS1, SIG1-2

p, (**T=AS3***) AS2, AS1, SIG2-3, SIG1-2

p, (**T=AS4***) AS3, AS2, AS1, SIG3-4, SIG2-3, SIG1-2

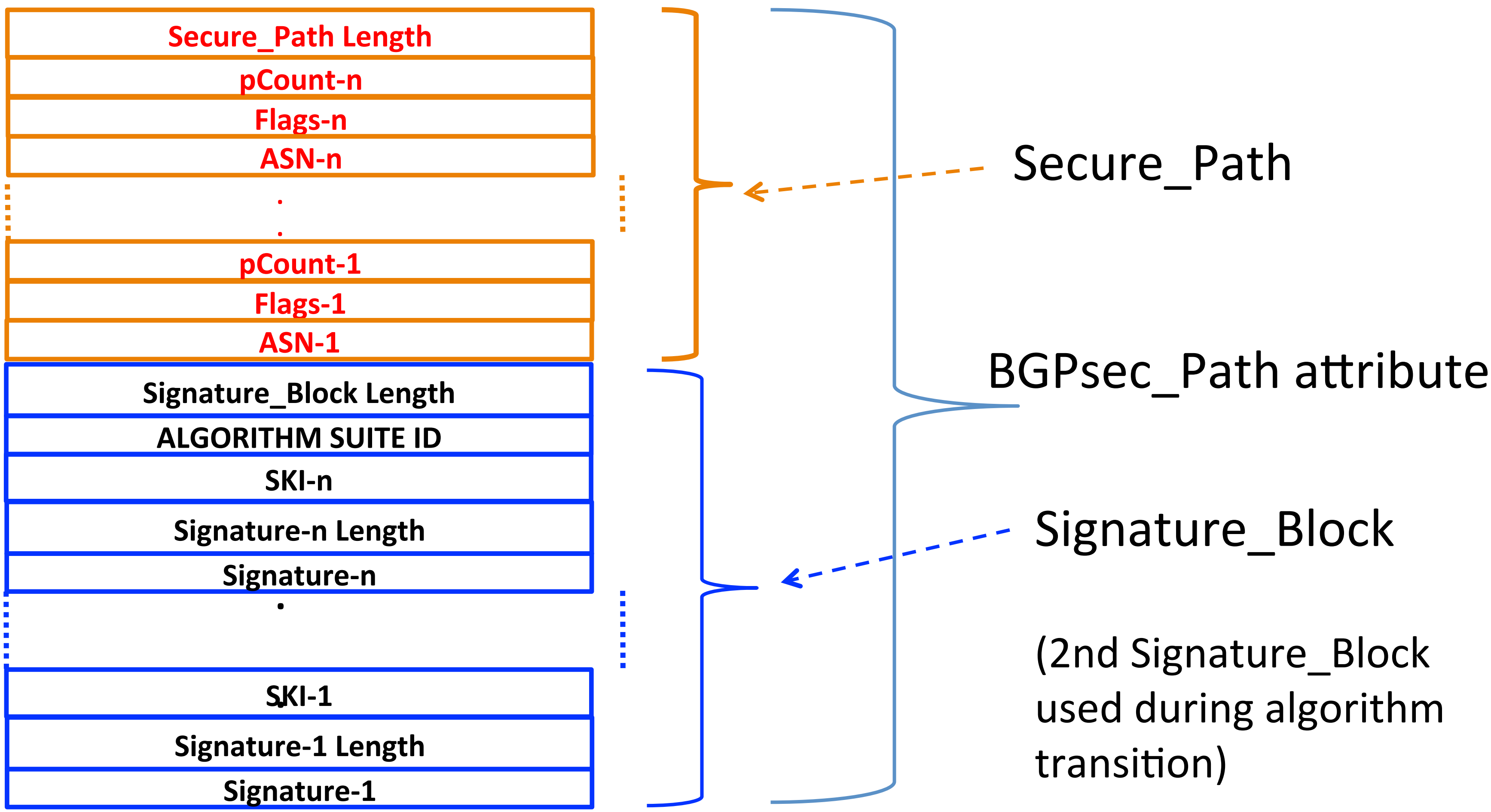
p, (**T=AS5***) AS4, AS3, AS2, AS1, SIG4-5, SIG3-4, SIG2-3, SIG1-2

Route Origin Authorization (ROA): (10.1.0.0/16, AS1, maxlength=18)
ROA is a signed object; stored in RPKI repository

* Next hop AS (**Target AS**) signed over but not carried in the

Note: For the precise BGPsec update format and details, see BGPsec specification
https://tools.ietf.org/html/draft-ietf-sidr-bgpsec-protocol-22

BGPsec_Path Attribute Format



- pCount is AS prepend count; pCount = 0 for transparent route server
- Flags: Confed_Flag set to indicate inter-AS hop within an AS confederation

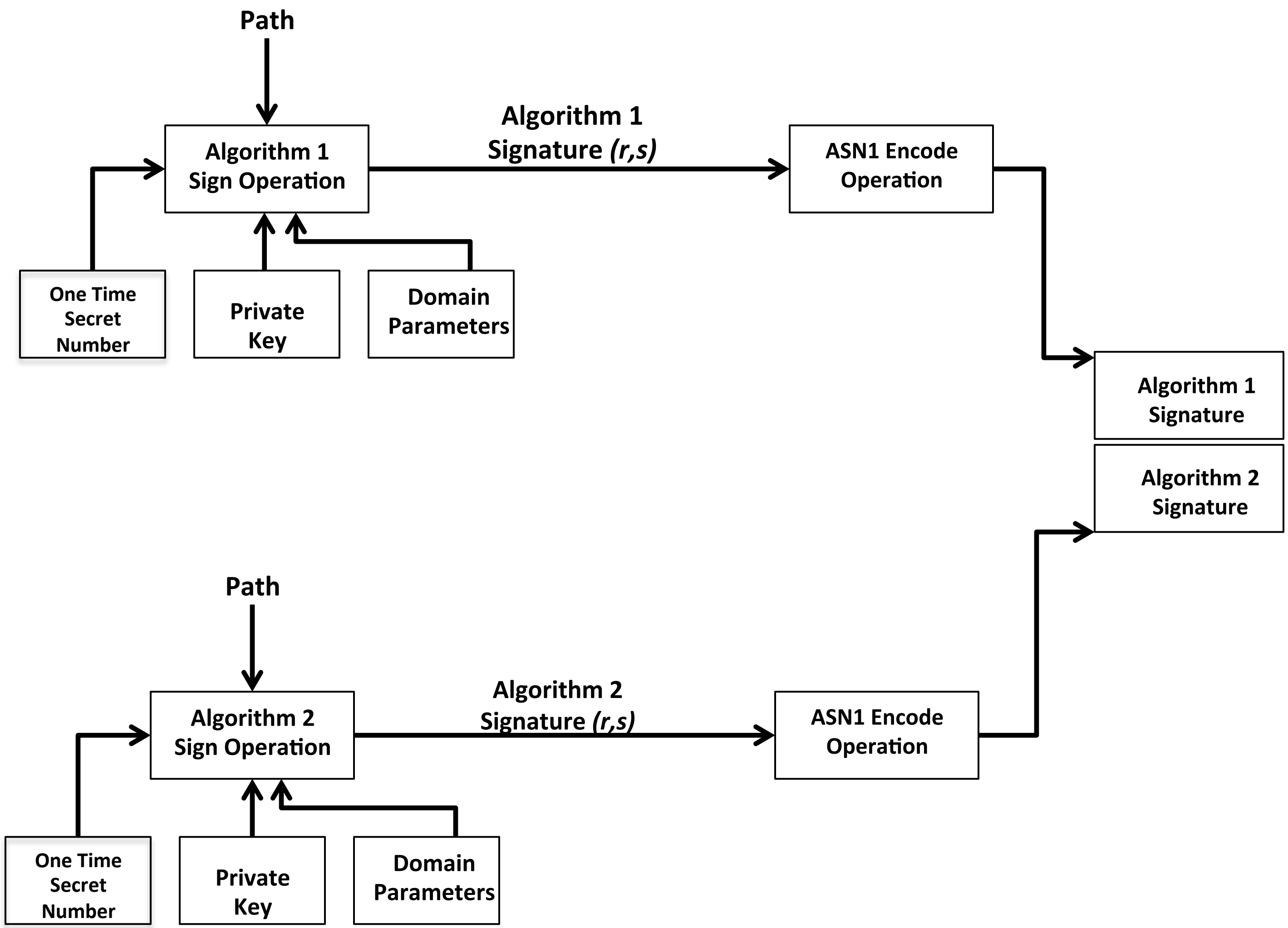
BGPsec Compute Resource Estimation

- BGP router receives approx. 680K prefixes [9] from each of its transit ISPs with an average path length of approx. <u>4 hops</u> [8]
- Internet contains about 55K ASs [8]
- A typical BGPsec router:
  - Verifies on average four signatures per supported algorithm, per update (two supported algorithms are allowed)
  - Generates one new signature per supported algorithm, per update

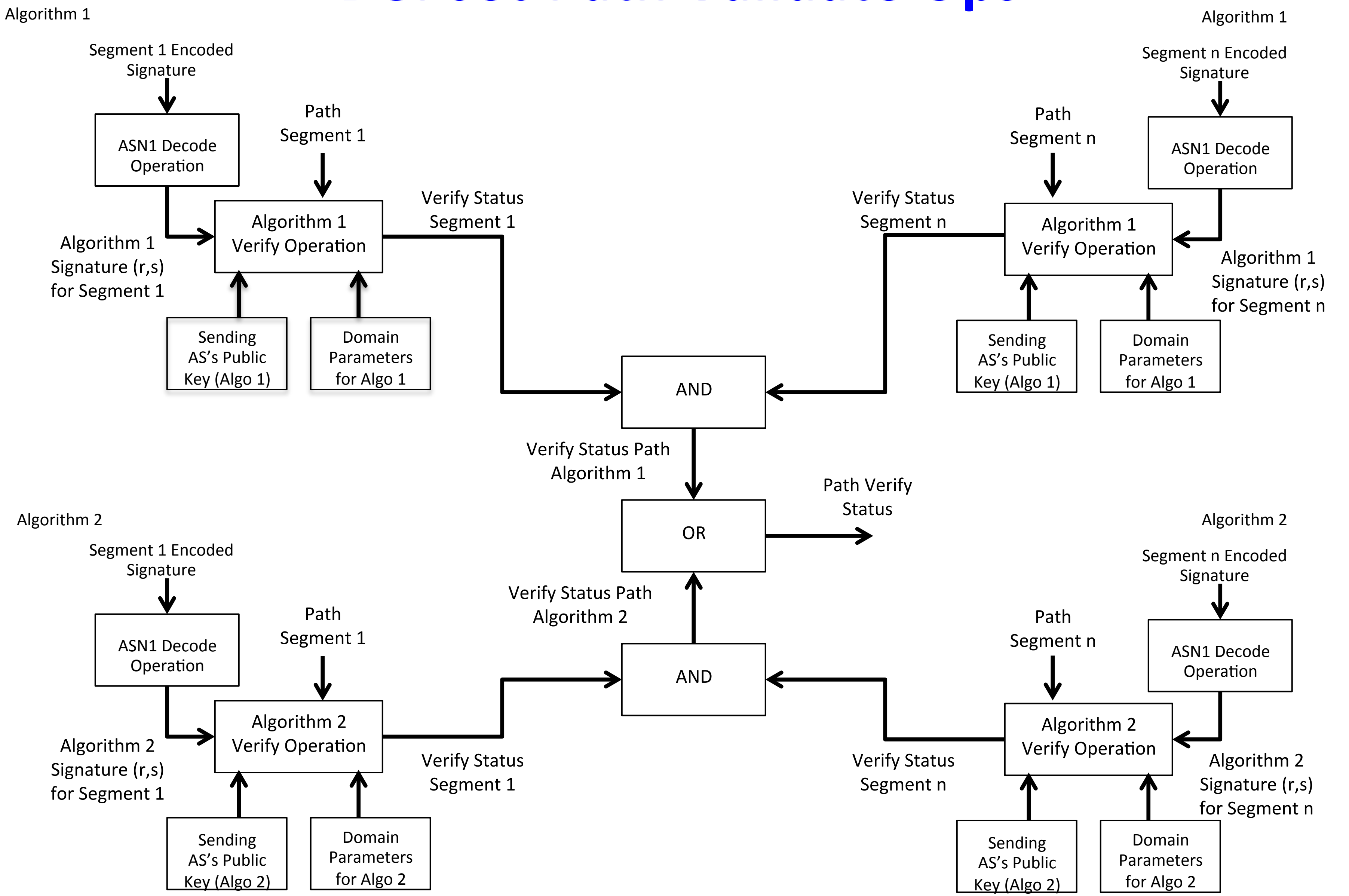
BGPsec Compute Resource Estimation (cont.)

- For Update Signing: Each AS will need at least one signing (private) key per supported algorithm…
  - Worst case: private key per router
  - Must be maintained securely (per NIST SP800-130, Confidentiality and Integrity)
- For Update Validation: an average of four public keys per supported algorithm need to be loaded
  - Public key retrieval includes public key database access and ECC Public-Key Validation (per NIST SP800-56A) per key
  - Caching helps

BGPsec Path Sign Ops



BGPsec Path Validate Ops



Optimizations

- Multi-level Optimizations are required to maximize performance
  - System Level Optimizations
    - Asynchronous operations across cores
    - Parallel multi-segment path verifications
    - High Integrity Public key management system
  - Algorithmic Optimizations
    - Early termination on Invalid segment (BGPsec Algorithm)
    - Pre-calculations for ECDSA sign Operation (ECDSA Algorithm)
  - Group Level Optimization
    - Ultra fast, secure Point Multiplication (e.g., Side Channel Attack (SCA) Resistant Fixed-base_NAF Windowing Method for Point Multiplication)
  - Field Level Optimizations
    - Special forms of domain parameters (e.g., Generalized Mersenne Primes)
    - Barrett Reduction modulo p and/or Montgomery w-by-w modulo

Optimizations must maintain and enhance the security of the implementation under all use-cases

Example Algorithmic Optimization

ECDSA Sign

1. Generate k *and* k^{-1}
2. Compute $R = kG$
3. Compute $r = x_R \bmod n$
4. Compute H = Hash (M)
5. Convert the bit string H to an integer e : where
   $e = \Sigma^{H}_{(i=1)} 2^{H-i} * b_i,$
6. $s = (k^{-1} * (e + d * r)) \bmod n$
7. Return (r, s)

Observation:

The most compute intensive ECDSA sign calculations do not have any dependency on the "message" to be signed

Options:

1. Pre-compute r and "safely" store
2. Asynchronously compute r on a different core
3. Proprietary methods

Considerations:

- Secure implementations are not trivial

Substantially reduces sign op latency

Example Group Level Optimizations

Pre-Calculation:

Take $(K_{d-1}, \ldots, K_1, K_0)_{2^w}$ as the base 2^w representation of k,

where $d = \lceil (m/w) \rceil$, then

$$kP = \Sigma^{d-1}_{(i=0)} K_i(2^{wi} P)$$

For each i from 0 to $d-1$,

pre-calculate j number of points, where

$j = (2^{w+1}-2)/3$ if w is even;

$j = (2^{w+1}-1)/3$ if w is odd

Evaluation:

INPUT: NAF(k), d, pT (Pointer to pre-computed data table)

OUTPUT: A = kP.

1. Evaluation: A←O
2. For i from 0 to $d-1$ do

   2.1 SafeSelect (Pi),

   use Ki=j to choose the appropriate $P[i][j]$ from Ptable (handle $-j$)

   2.2 A←A + Pi
3. Return(A)

Traditional Right to Left Binary Method for Point Multiplication

- Evaluation time: $(0.5m)$pA + (m)pD
- P-256 Eval. time: **128pA + 256pD**
- Not SCA resistant

Fixed-base_NAF Windowing Method for Point Multiplication

- Evaluation time: $(0.5m)$pA + (m)pD
- P-256 Eval. time: **~64pA**
- SCA resistant

m: number of bits; pA: multi-precision point addition; pD: multi-precision point double

*tara*EcCRYPT™ Performance

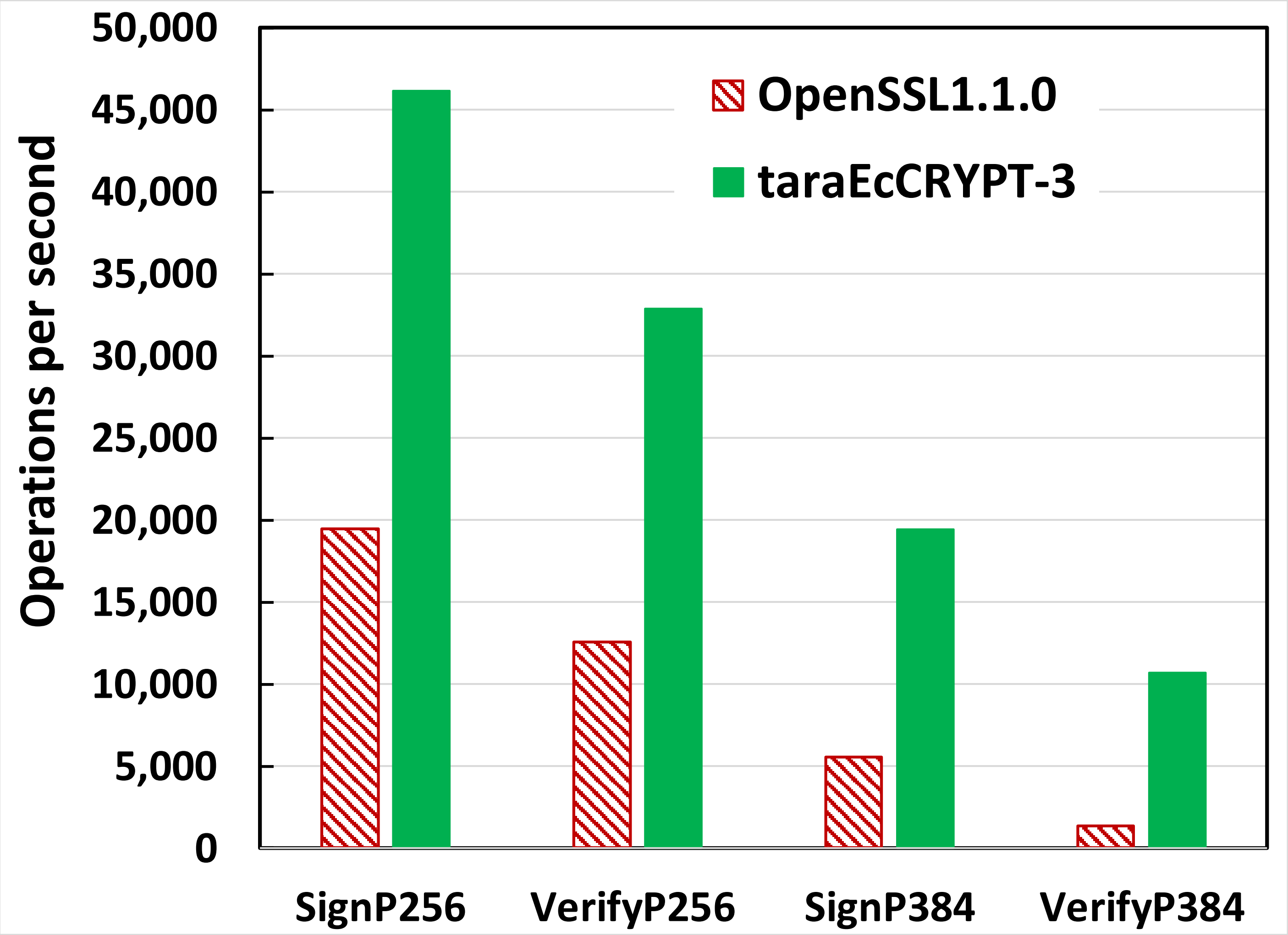
- One of the fastest available libraries for ECDSA P-256 and P-384 operations
- Thread-safe dynamic library
- Single core performance numbers are captured with a standalone utility to show the best possible rates

ECDSA Ops	*tara* EcCRYPT-3 P-256 Rate (ops/sec)	*tara* EcCRYPT-3 P-384 Rate (ops/sec)
Sign Operation with provided Hash	**46,191.83**	**19,433.85**
Sign Op including Hash gen Op	**40,076.27**	**18,163.09**
Sign Op including Hash gen Op, but using pre-calculated random-number	**48,257.51**	**21,287.04**
Verify Op with provided Hash	**32,895.25**	**10,954.57**
Verify Op including Hash gen Op	**29,521.49**	**10,706.65**

System used for all test results:

- Intel® Xeon® CPU E3-1285 v4 at 3.5GHz; 16GB Memory; Centos 7; gcc 5.2
- Test results on single core; Hyper-threading and Turbo features turned off; Nominal message size 1024 bytes

openSSL vs. *tara*EcCRYPT™ Performance



(With provided hash in all cases)

System used for all test results:

- Intel® Xeon® CPU E3-1285 v4 at 3.5GHz; 16GB Memory; Centos 7; gcc 5.2
- Test results on <u>single core</u>; Hyper-threading and Turbo features turned off

taraBGPsec™ Path Sign Performance

- Measured operation includes:
  - ➢ Assembly of the BGPsec Path data to be signed
  - ➢ Hash of path data
  - ➢ Execution of Path_Signature operation using ECDSA P-256 or P-384
  - ➢ ASN1 encode of signature(s)

- On a single core using *tara*EcCRYPT Sign with pre-calculated random-number:
  - ➢ P256 Path sign performance is over 40k signatures/sec
  - ➢ P384 Path sign performance is over 20k signatures/sec

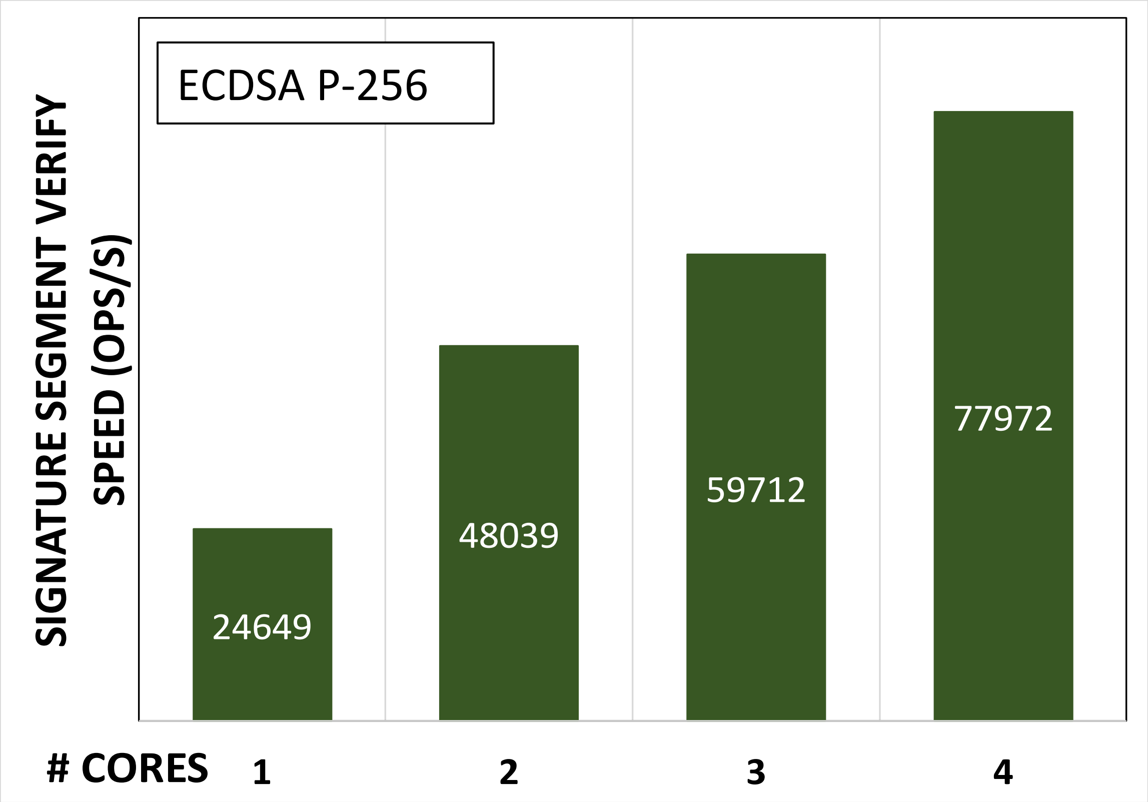
- Given current performance levels, multi-core parallelization may not be needed to sustain a high number of signature operations.
  - ➢ However, proper random numbers can be pre-calculated asynchronously on any available core

Same system utilized to generate the test results as on slide #15.

taraBGPsec™ Path Verify Performance

Measured operation includes:

- BGPsec related parsing of update packets
- Fetching public keys with assured integrity
- Execution of Path_Segment_Verify operations (*tara*VerifyParallel)
  - ✓ Early termination if any Segment of the path is found to be Invalid
  - ✓ Two signature algorithms (P-256 and P-384) can be supported



Same system utilized to generate the test results as on slide #15, except the number of cores is varied.

BGPsec Contribution to Convergence Time

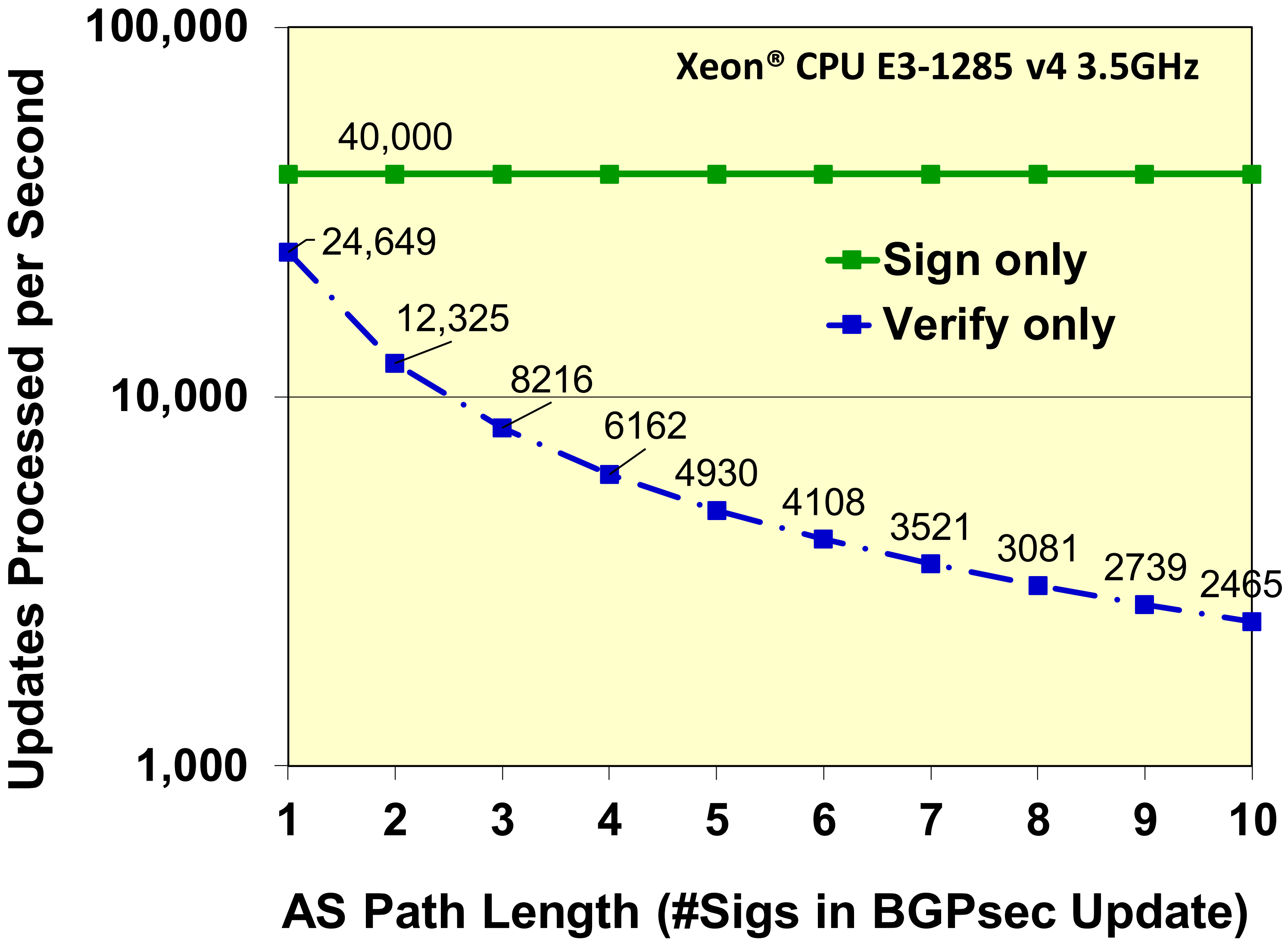
BGPsec update processing is **Additive** to Traditional BGP processing

Convergence time

Traditional BGP	BGPsec
• Best path selection • Peering policies • Route filtering • RIB management, etc.	• Parsing BGPsec update • Data assembly for hashing • Fetching public keys • Signature verification • Signing to peers

We focus here on the incremental CPU cost due to BGPsec

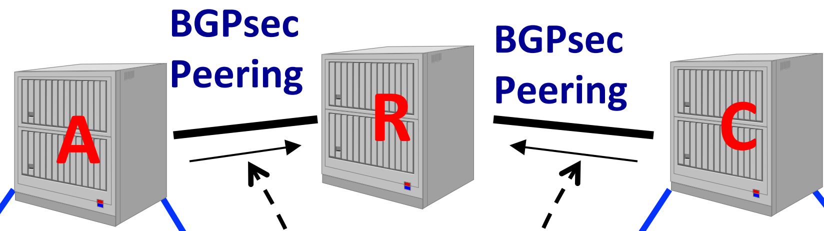
BGPsec Verify / Sign Speed (Updates/s)



- taraBGPsec
- Xeon® CPU E3-1285 v4 3.5GHz (using only one core)

Validation Cost Model

- taraBGPsec
- Xeon® CPU E3-1285 v4 3.5GHz (using only one core)



ISP A and A's Customer Cone (Large Global ISP)

CPU Time on R if Session to A is Reset

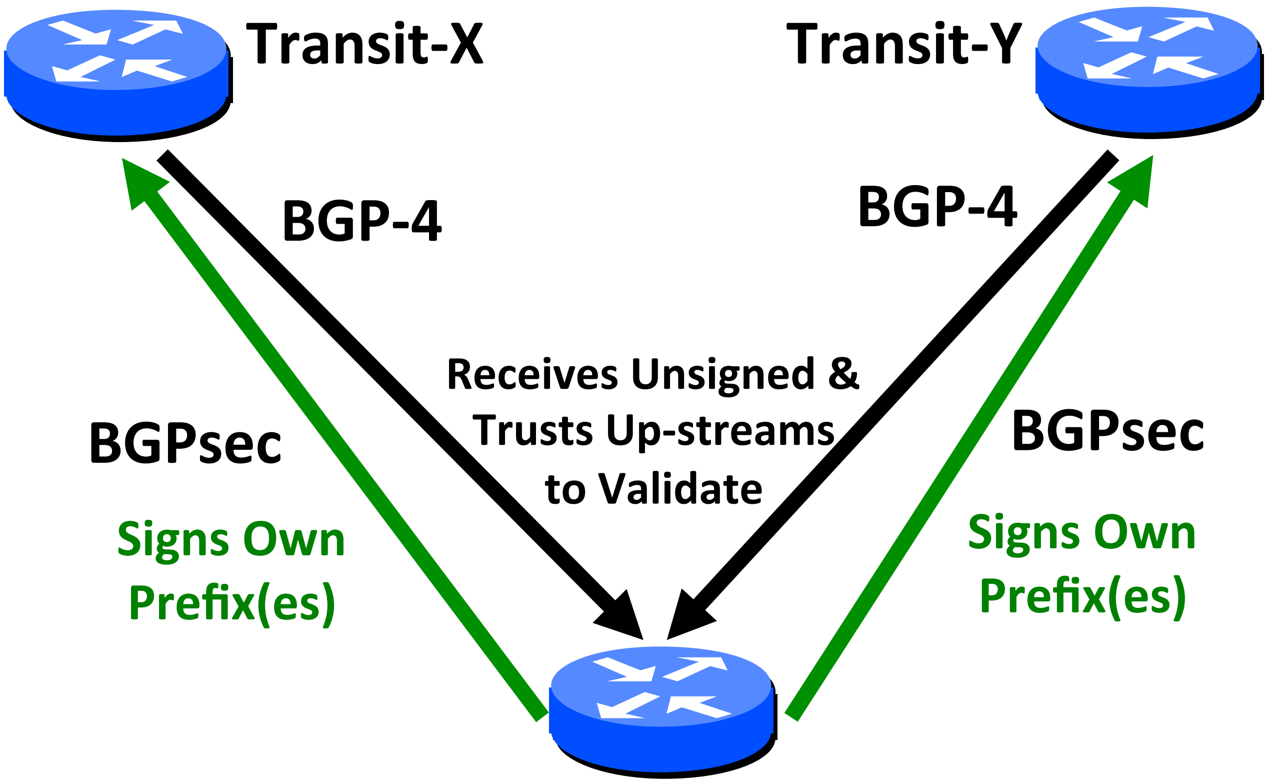
AS path length	# Prefixes announced	Processing time (sec)
1	1353	0.0549
2	21586	1.7515
3	6820	0.8301
4	1627	0.2640
5	942	0.1911
6	45	0.0110
7	14	0.0040
8	6	0.0019
Total (seconds)		3.11

ISP C and C's Customer Cone (Large Global ISP)

CPU Time on R if Session to C is Reset

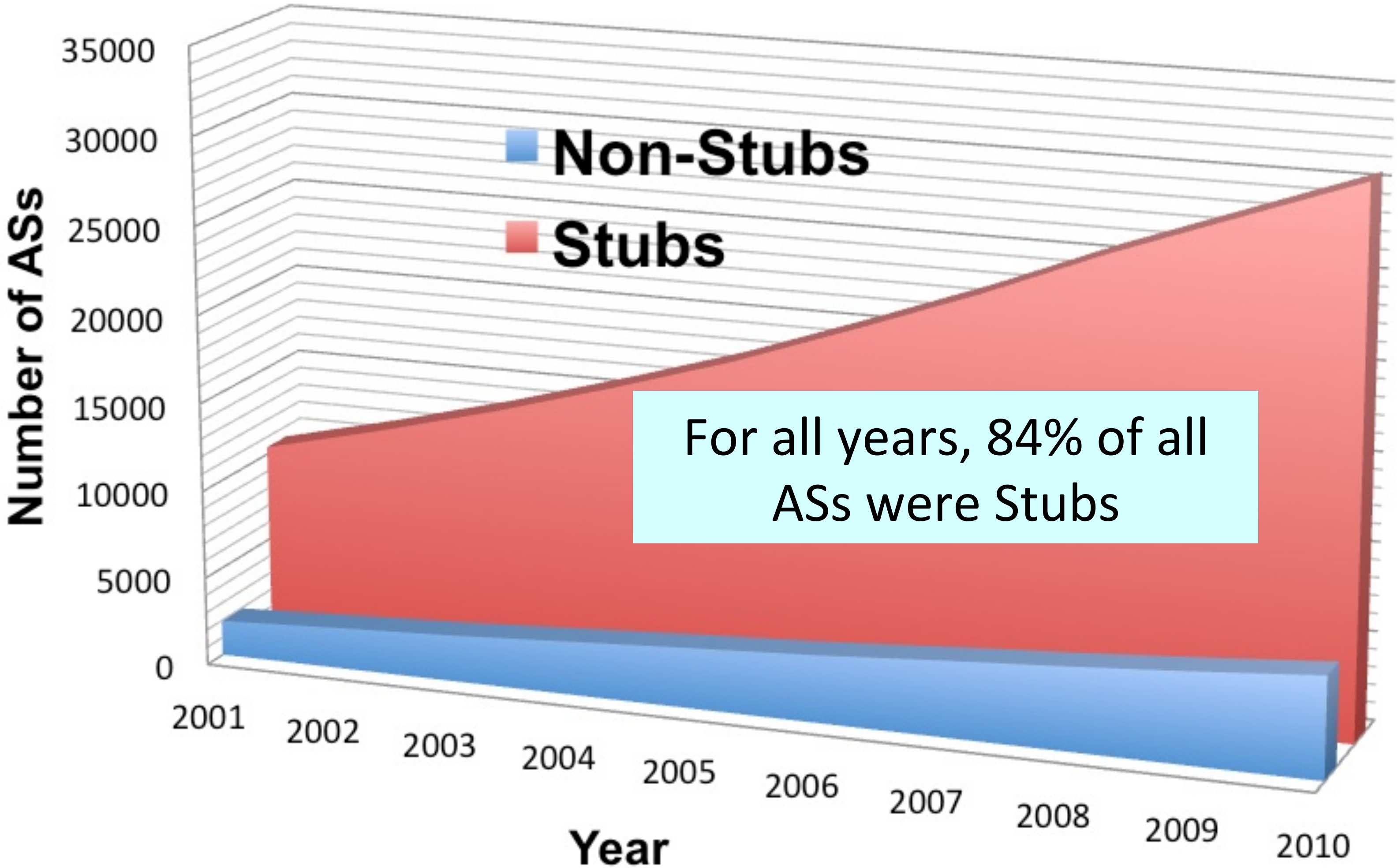
AS path length	# Prefixes announced	Processing time (sec)
1	620	0.0252
2	16028	1.3005
3	9434	1.1482
4	2922	0.4742
5	435	0.0882
6	46	0.0112
7	15	0.0043
8	27	0.0088
9	1	0.0004
Total (seconds)		3.06

Need not Sign To Stubs

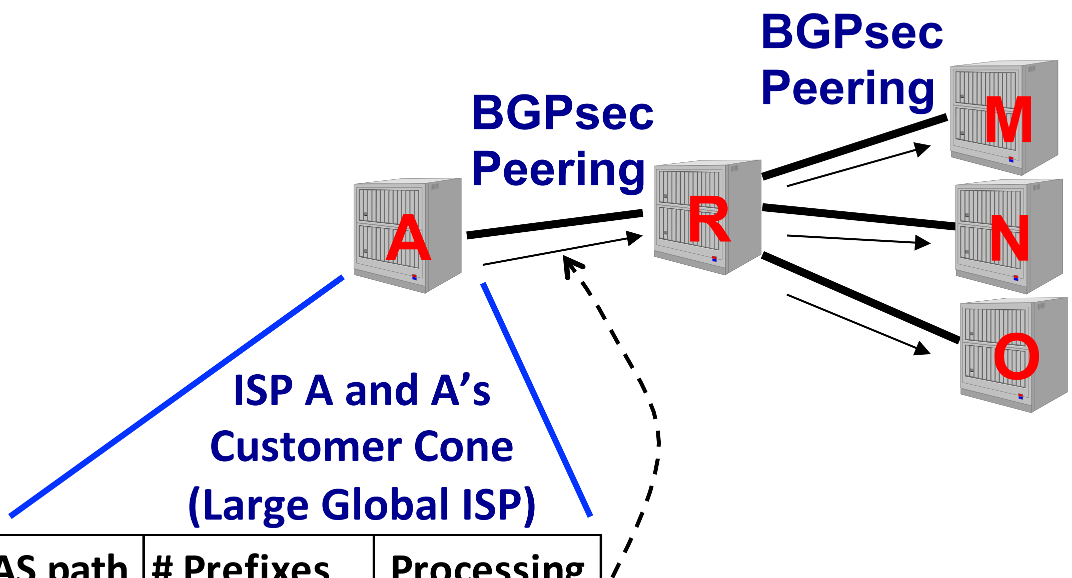


Only needs to have appropriate ROAs registered
and own Private Key;
No other crypto or RPKI data
No Hardware Upgrade!!

What Fraction are Stub ASs?



CPU Cost for Validation and Signing



- R peers with 3 non-stub BGPsec peers
- R's other peers are stub ASs

AS path length	# Prefixes announced	Processing time (sec)
1	1353	0.16
2	21586	3.37
3	6820	1.34
4	1627	0.39
5	942	0.26
6	45	0.01
7	14	0.01
8	6	0.00
Total (seconds)		5.54

CPU workload on R, including Validation & Signing, if session to A is reset.

- taraBGPsec
- Xeon® CPU E3-1285 v4 3.5GHz (using only one core)

Conclusions

- Industry leading high performance BGPsec implementation architected and algorithmically optimized
- Other factors expected to contribute to efficient BGP/BGPsec convergence in future:
  - Multi-threading and use of multiple cores in route processors
  - Update processing optimizations, e.g. caching results of verification
  - Enhanced Graceful Restart [10]

Thank you.

Questions?

References

1. M. Lepinski and K. Sriram, “BGPsec Protocol Specification,” https://datatracker.ietf.org/doc/draft-ietf-sidr-bgpsec-protocol/
2. M. Adalier, “Efficient and Secure Elliptic Curve Cryptography Implementation of Curve P-256,” NIST Workshop on ECC Standards, http://csrc.nist.gov/groups/ST/ecc-workshop-2015/papers/session6-adalier-mehmet.pdf
3. OpenSSL: Cryptography and SSL/TLS Toolkit, https://www.openssl.org/
4. R. Gagliano, K. Patel, and B. Weis, "BGPsec Router Certificate Rollover", draft-ietf-sidr-bgpsec-rollover-05 (work in progress), March 2016. https://tools.ietf.org/html/draft-ietf-sidr-bgpsec-rollover-06
5. K. Sriram, D. Montgomery, B. Dickson, K. Patel, and A. Robachevsky, "Methods for Detection and Mitigation of BGP Route Leaks," draft-sriram-idr-route-leak-detection-mitigation-04, July 2016. https://datatracker.ietf.org/doc/draft-ietf-idr-route-leak-detection-mitigation/
6. K. Sriram and Randy Bush, "Estimating CPU Cost of BGPSEC on a Router," presented at the RIPE 63, November 2011. https://ripe63.ripe.net/presentations/127-111102.ripe-crypto-cost.pdf
7. K. Sriram, D. Montgomery, and R. Bush, "RIB Size and CPU Workload Estimation for BGPSEC," Presentation at the IETF-91 Joint IDR/SIDR WG Meeting, November 2014. https://www.ietf.org/proceedings/91/slides/slides-91-idr-17.pdf
8. Measurements on AS paths, http://bgp.potaroo.net/as6447/
9. Measurements on routed IPv4 and IPv6 prefixes and growth rates, http://bgp.potaroo.net/v6/v6rpt.html
10. K. Patel, E. Chen, R. Fernando, and J. Scudder, “Accelerated Routing Convergence for BGP Graceful Restart,” https://datatracker.ietf.org/doc/draft-ietf-idr-enhanced-gr/
11. A. Lambrianidis and E. Nguyenduy, “Route server implementations performance,” 20th Euro-IX Forum, Amsterdam, NL, April 2012. https://ams-ix.net/downloads/ams-ix-route-server-implementations-performance.pdf

Backup Slides

Example Field Level Optimizations

- Multi-precision regular/constant time add and subtract modulo prime ops are best implemented in x86-assembly
  - Any Carry or Borrow is easily detected
  - Handled by instructions such as “adcq” and “sbbq”
- Optimized multi-precision multiply and square operations are a must for high performance

Traditional 64-bit multiply in x86

```
mov OP, [pB+8*0]
mov rax, [pA+8*0]
mul OP
add R0, rax
adc rdx, 0
mov TMP, rdx
mov pDst, R0
mov rax, [pA+8*1]
mul OP
mov R0, rdx
add R1, rax
adc R0, 0
add R1, TMP
adc R0, 0
mov rax, [pA+8*2]
mul OP
mov TMP, rdx
add R2, rax
adc TMP, 0
add R2, R0
adc TMP, 0
...
```

64-bit multiply with Broadwell Inst.

```
xor rax, rax
mov rdx, [pB+8*0]

mulx T1, T2, [pA+8*0]
adox R0, T2
adcx R1, T1
mov pDst, R0

mulx T1, R’0, [pA+8*1]
adox R’0, R1
adcx R2, T1
mulx T1, R’1, [pA+8*2]
adox R’1, R2
adcx R3, T1
...
```