Digital Signature Algorithm Implementations

Leah Chatkeonopadol & Smruthi Manjunath

University of California, Santa Barbara

chatkeon@cs.ucsb.edu
smanjunath@cs.ucsb.edu

CS290G APPLIED CRYPTOGRAPHY

DSA vs. ECDSA

- DSA requires larger parameters for the same level of security as ECSA
- ECDSA takes less memory
- ECDSA scales more efficiently than DSA in terms of security level
- ECDSA—easier to find inverse of a point on a curve vs. finding modular inverse

DSA Algorithm

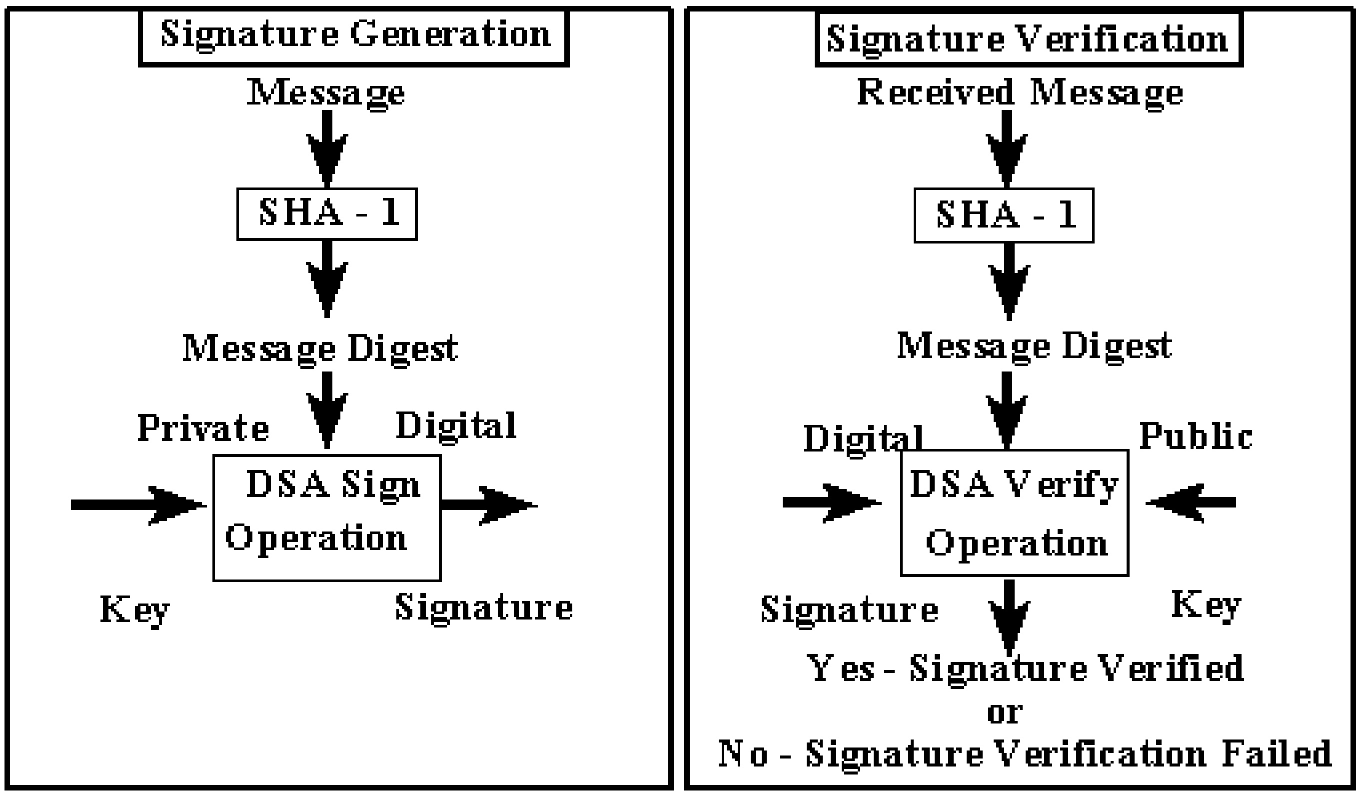


Figure 1: Using the SHA-1 with the DSA

SHA-1

- Pad message $\rightarrow$ [message][1][0's][len]
  - len $=$ length of original message as a 64-bit $\#$
  - Entire padded message $=$ 512*n bits

- Split message into 512-bit blocks (16 words, each 32-bits long)

- For each block: perform an 80-step iteration
  - Uses 2 buffers: A, B, C, D, E and H_0, H_1, H_2, H_3, H_4
  - Start with specified constants

- At the end, H_0, H_1, H_2, H_3, H_4 contain the 160-bit output

DSA: Signature Generation

- Hashed message (SHA-1): $H(m)$
- Public key: (p, q, g, y)
  - p, q are co-prime
  - $y = g^x \bmod p$
- Private key: x
- Random value per message: k
- Calculate $r = (g^k \bmod p) \bmod q$
- Calculate $s = k^{-1}(H(m) + xr) \bmod q$
- Signature: (r, s)

DSA: Signature Verification

- Hashed message (SHA-1): $H(m)$
- Signature: (r, s)
- Calculate $w = s^{-1}$ mod q
- Calculate $u_1 = (H(m) * w)$ mod q
- Calculate $u_2 = (r * w)$ mod q
- Calculate $v = ((g^{u_1} y^{u_2})$ mod $p)$ mod q
- If $(v == r)$: VALID

ECDSA: Signature Generation

- Hashed message (SHA-1): $H(m)$
- Parameters: P, G, n; Public key: Q; Private key: d, where $[d]G = Q$
- Random value per message: k
- Calculate point $(x_1, y_1) = [k]G$
- Calculate $r = x_1 \bmod n$
- Calculate $s = (k^{-1} * (z + r * d)) \bmod n$
- Signature: (r, s)

ECDSA: Signature Verification

- Hashed message (SHA-1): $H(m)$
- Parameters: P, G, n; Public key: Q
- Signature: (r, s)
- Calculate $w = s^{-1}$ mod n
- Calculate $u_1 = (z * w)$ mod n and $u_2 = (r * w)$ mod n
- Calculate point (x_1, y_1) =$[u_1]G$+$[u_2]Q$
- If $(r == x_1$ mod $n)$: VALID

Implementation

- Python running on a Mac (2.4 GHz Intel Core 2 Duo)
- DSA
  - Binary method, no other optimizations
  - Extended Euclidean algorithm for finding inverses
- ECDSA
  - Recoding + binary method with subtractions
  - Extended Euclidean algorithm for finding inverses
  - Four different coordinate systems

Coordinate Systems

- Affine: x, y
- Projective Jacobian: $X, Y, Z \rightarrow x = X/Z^2, y = Y/Z^3$
- Projective Standard: $X, Y, Z \rightarrow x = X/Z, y = Y/Z$
- Modified Jacobian: $X, Y, Z, T \rightarrow x = X/Z^2, y = Y/Z^3 (T = a * Z^4)$

Parameters

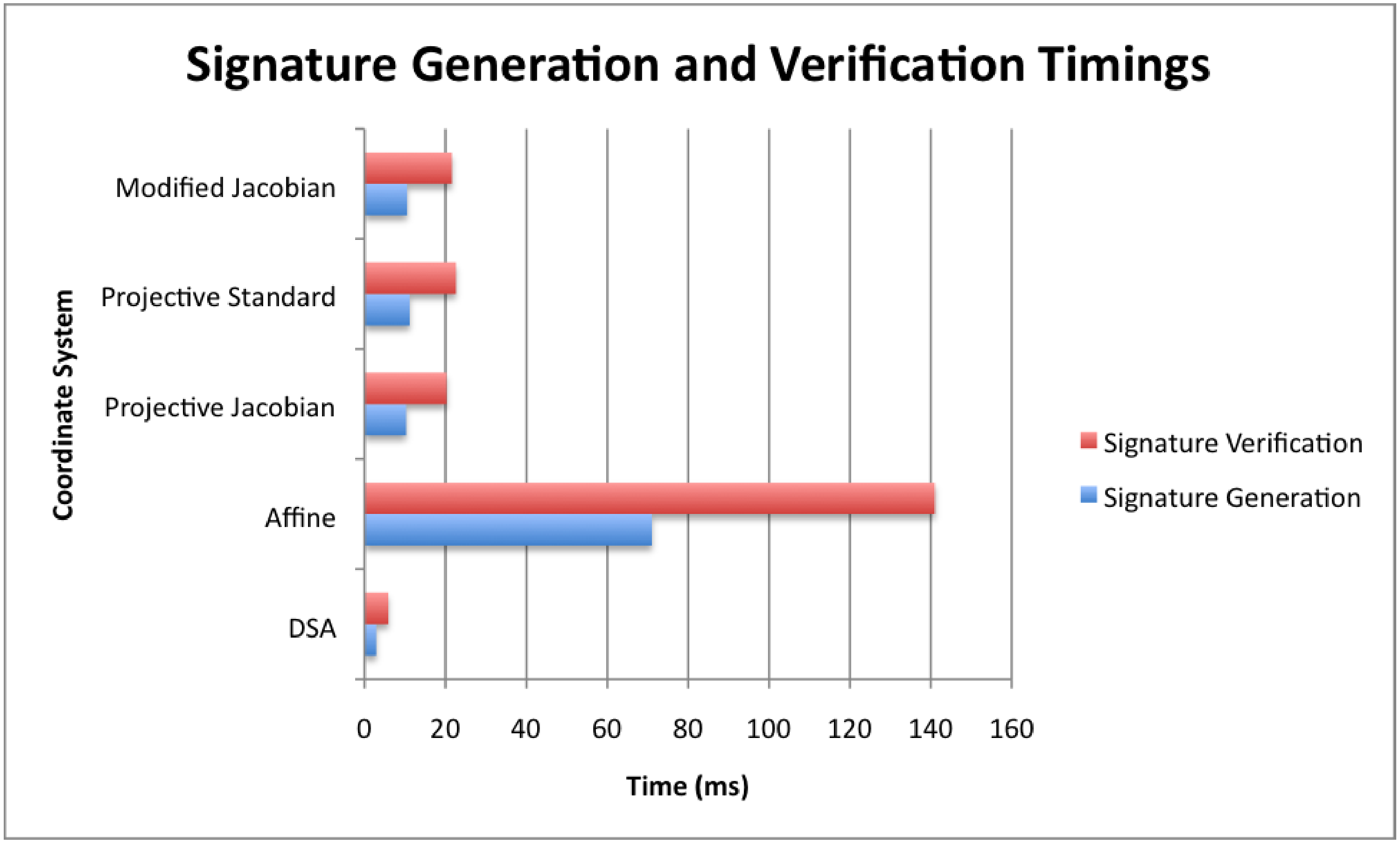
- DSA: http://csrc.nist.gov/groups/ST/toolkit/documents/dss/Examples-1024bit.pdf

- ECDSA: NIST curve P-224, http://tools.ietf.org/html/draft-pornin-deterministic-dsa-01#appendix-A.2.4

Demo

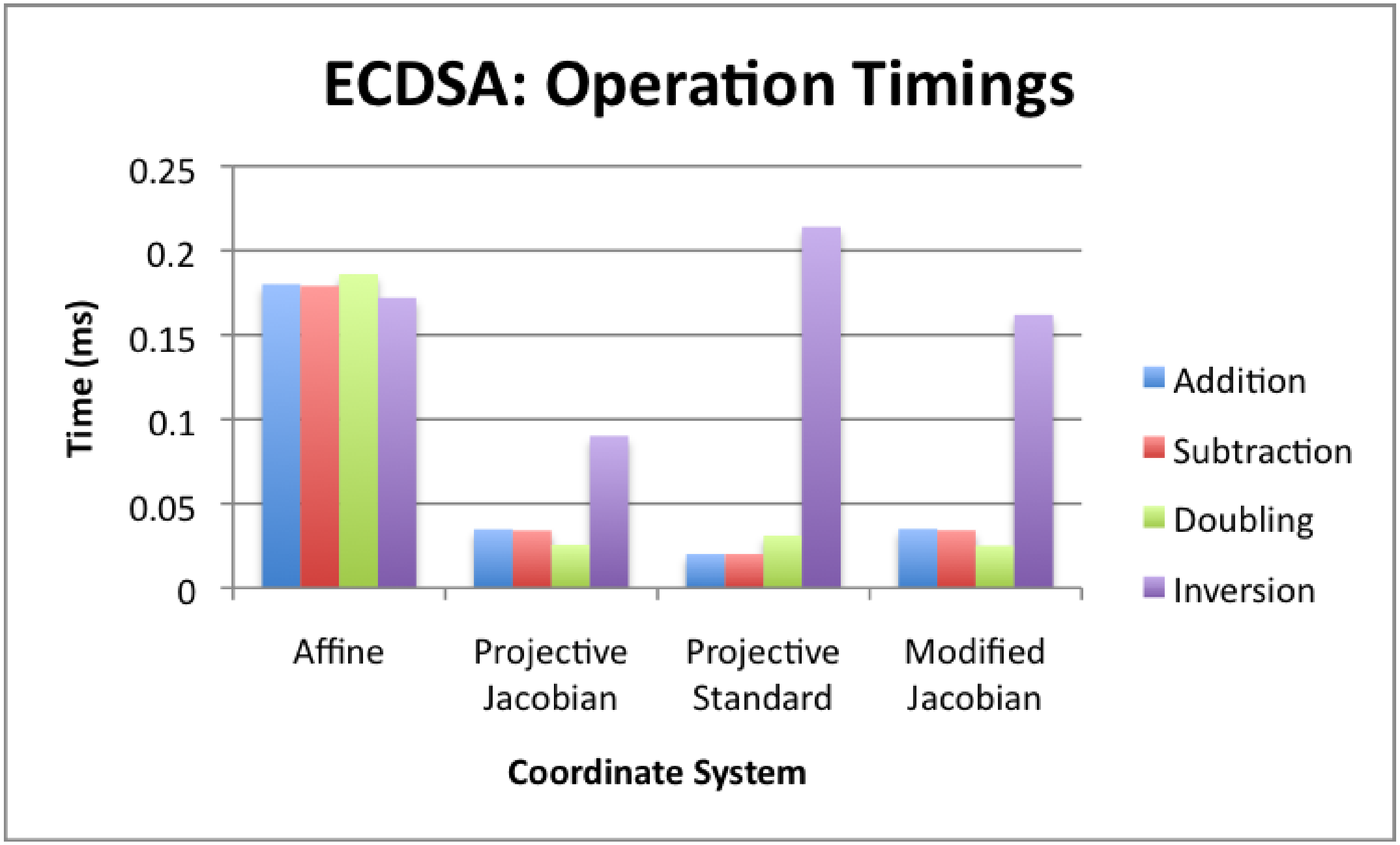
- DSA: Signature Generation and Verification
- ECDSA: Signature Generation and Verification

Results



Signature Generation and Verification Timings
Coordinate System
Modified Jacobian
Projective Standard
Projective Jacobian
Affine
DSA
0
20
40
60
80
100
120
140
160
Time (ms)
Signature Verification
Signature Generation

Results



ECDSA: Operation Timings
0.25
0.2
0.15
0.1
0.05
0
Time (ms)
Affine
Projective Jacobian
Projective Standard
Modified Jacobian
Coordinate System
Addition
Subtraction
Doubling
Inversion

Results

Coordinate System	Addition	Subtraction	Doubling	Inversion	Total
DSA	77	0	158	0	235
Affine	94	113	671	878	1756
Projective Jacobian	94	113	671	6	884
Projective Standard	94	113	671	3	881
Modified Jacobian	94	113	671	3	881

Table 1: Operation Counts for Different Coordinate Systems (and DSA)

Conclusion

- Hash function takes a lot of time ($\sim$100x)
- Projective Jacobian was the best coordinate system
  - Projective Jacobian, projective Standard, and modified Jacobian were comparable
  - Affine is much slower because of the inversions
- Implementation details have a large effect on performance
  - Example: Fermat's theorem vs. Extended Euclidean
  - Example: Projective Jacobian—division by 2
  - Python may optimize modulation

References

- http://csrc.nist.gov/groups/ST/toolkit/documents/dss/Examples-1024bit.pdf
- http://tools.ietf.org/html/draft-pornin-deterministic-dsa-01#appendix-A.2.4
- http://www.itl.nist.gov/fipspubs/fip180-1.htm
- FIPS 186-3
- http://www.hyperelliptic.org/EFD/