iHTTP: Efficient Authentication of Non-Confidential HTTP Traffic

Jason Gionta, Peng Ning
North Carolina State University

Xiaolan Zhang
IBM T.J. Watson Research Center

Background

- Website owners rely on HTTP for revenue
  - Advertising
    - $14.9 billion in first half of 2011, 23 % increase*
- Authenticity of HTTP data still important!
  - AD rewriting
    - Ex. rewrite ads incoming over public WiFi
  - AD injection
- Solved by enabling data authentication

*The Interactive Advertising Bureau

Motivation: Problem

- How to enable authentication of HTTP data?
  - Less costly than HTTPS
  - Supports network caching
    - Enables distributed model of HTTP
  - Data authentication and integrity
    - Clients can determine if data has been tampered
  - Data freshness authentication
    - Clients can verify that data is not being replayed

Previous Research

- Enabling authentication of HTTP data
  - SSL Splitting [SSYM 2003]
    - Caches are modified and trusted
  - Keyed Hashing [ICCN 2011]
    - Key exchange and management required – SSL/TLS
  - Signature based HTTP integrity (SBHI)
    - Framework for enabling authentication of HTTP response data
    - No SSL
    - No modifications to network caches

Previous Research cont.

- Signature Based HTTP Integrity
  - SINE [NPSec 2009]
    - Enables progressive rendering
      - Progressive Authenication
    - Cache signatures to amortize costs
    - No data freshness assurance
  - HTTPi [WWW 2012]
    - Enables support for HTTP/1.1 chunked encoding
    - Ensures data freshness authentication
      - Sign authenticator with updated timestamps

Signature Based HTTP Integrity

- Problems with existing SBHI approaches
  - Insecure - SINE
    - Susceptible to replay attacks
      - Cannot authenticate freshness
  - Inefficient - HTTPi
    - Must sign each response – COSTLY!
  - Clients must verify each signature
    - Possibly hundreds of verifications per page
- Need for efficient secure HTTP authentication

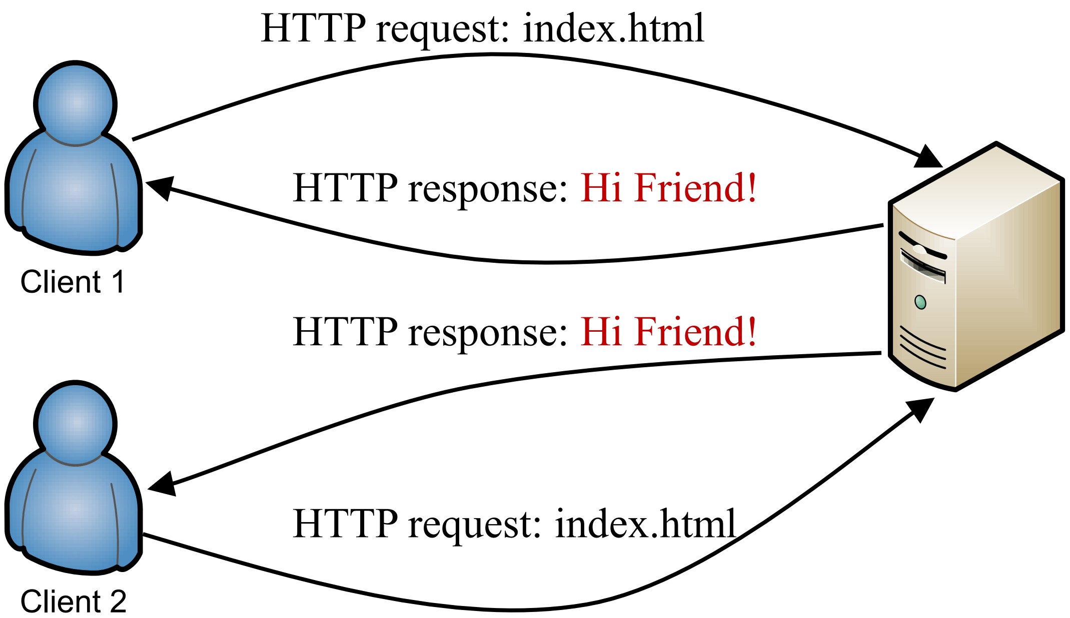
Our Approach: iHTTP

- Authentication and integrity of HTTP response data
  - SBHI based
- Enable data freshness authentication
  - Authenticated timestamp for signature
- Lightweight and efficient
  - Dynamically update timestamp without signing
  - Assist clients in signature authentication
- Requires no modification to network caches

iHTTP: Definition

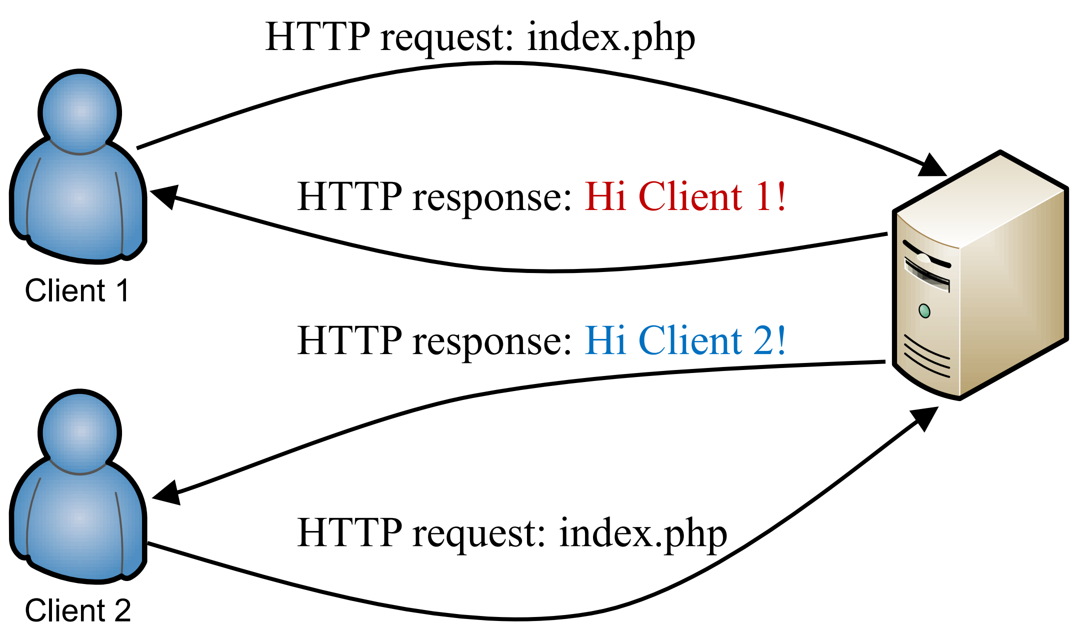
- Observation
  - Two categories of HTTP data
    - Client-Static
    - Client-Unique

iHTTP: Client-Static



Many to one relationship: many clients one response

iHTTP: Client-Unique



Many to many relationship: many clients many response

iHTTP: Assumptions and Threat Model

- Assumptions
  - Clients and servers are loosely synchronized
  - Response data is ***Client-Static*** and ***non-confidential***
- Threat model
  - Intercept, modify, store, replay all data between the client and server
  - Slow down data delivery for reasonable amounts of time

iHTTP: Authenticator generation

- Authenticator content
  - Authenticator metadata
  - Response metadata
  - Message body hashes

$A.t$ – generation timestamp
$A.e$ – expiration
$A.u$ – requested URL
$A.l$ – length of authenticated data

$$Sign_k\{H(A.t \mid A.e \mid A.u \mid A.l \mid HTTP.Headers \mid HTTP.Content)\}$$

- Generation occurs:
  - New response data is observed for a URL
  - Authenticator expires

iHTTP: Authenticator generation cont.

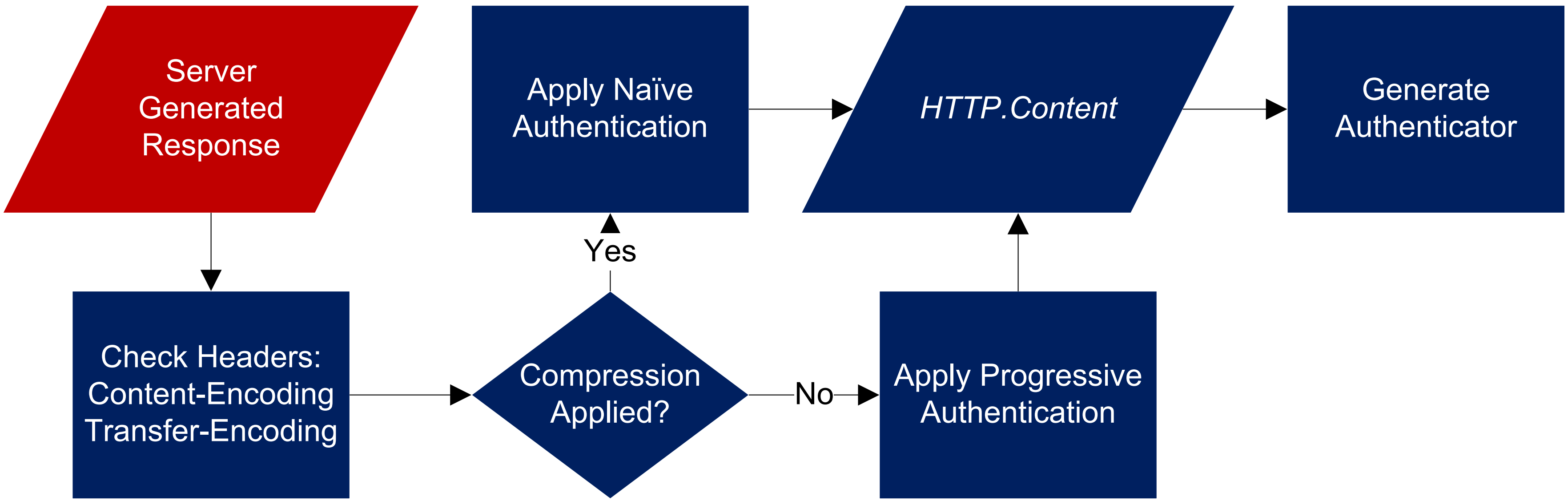
- *HTTP.Headers*
  - Protect headers not modifiable by caches
  - Consists of End-to-end headers
    - Defined by HTTP/1.1
      - Non-connection specific headers
    - Exception
      - Length header which may be changed by caches
      - *A.l* provides size of data protected by authenticators

iHTTP: Authenticator generation cont.

- *HTTP.Content* generation
  - Used to verify the integrity of the message body
  - Leverages Naïve and Progressive Authentication
    - Observation
      - Compressed data must be buffered on both the server and client
      - Progressive Authentication not necessary for compressed data
    - Result
      - Naïve is optimal for compressed data
      - Progressive Authentication beneficial for non-compressed data

iHTTP: Adaptive Data Handling

- Rule for generation of *HTTP.Content*



  – Hash operations
    - Naïve Authentication: $O(1)$
    - Progressive Authentication: $O(n)$

:Server
:iHTTP

iHTTP: Freshness Authentication

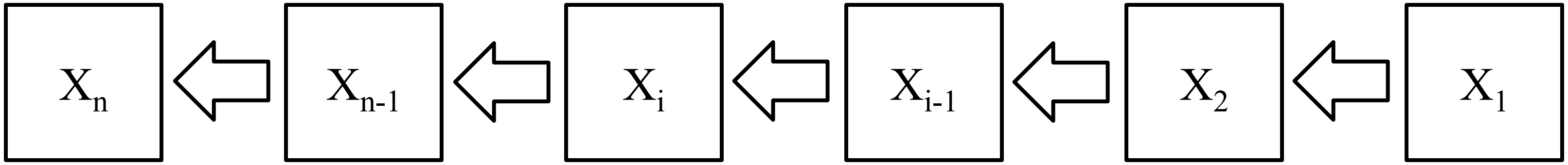
- Expiration *A.e* allows long lived authenticators
  - When response content does not change
  - Bypasses signing; allows authenticator caching
    - Significant performance benefit
- Problem: *A.e* is static after signing
  - Attackers can replay data that has not yet expired
  - Removing *A.e* requires signing each response

iHTTP: Efficient Freshness Authentication

- How to enable data freshness efficiently?
  - Sliding-Timestamps
    - Decouple data freshness and authenticator generation
      - Freshness not tied to authenticator timestamp
    - Enables caching of authenticators
      - Amortizes signing costs

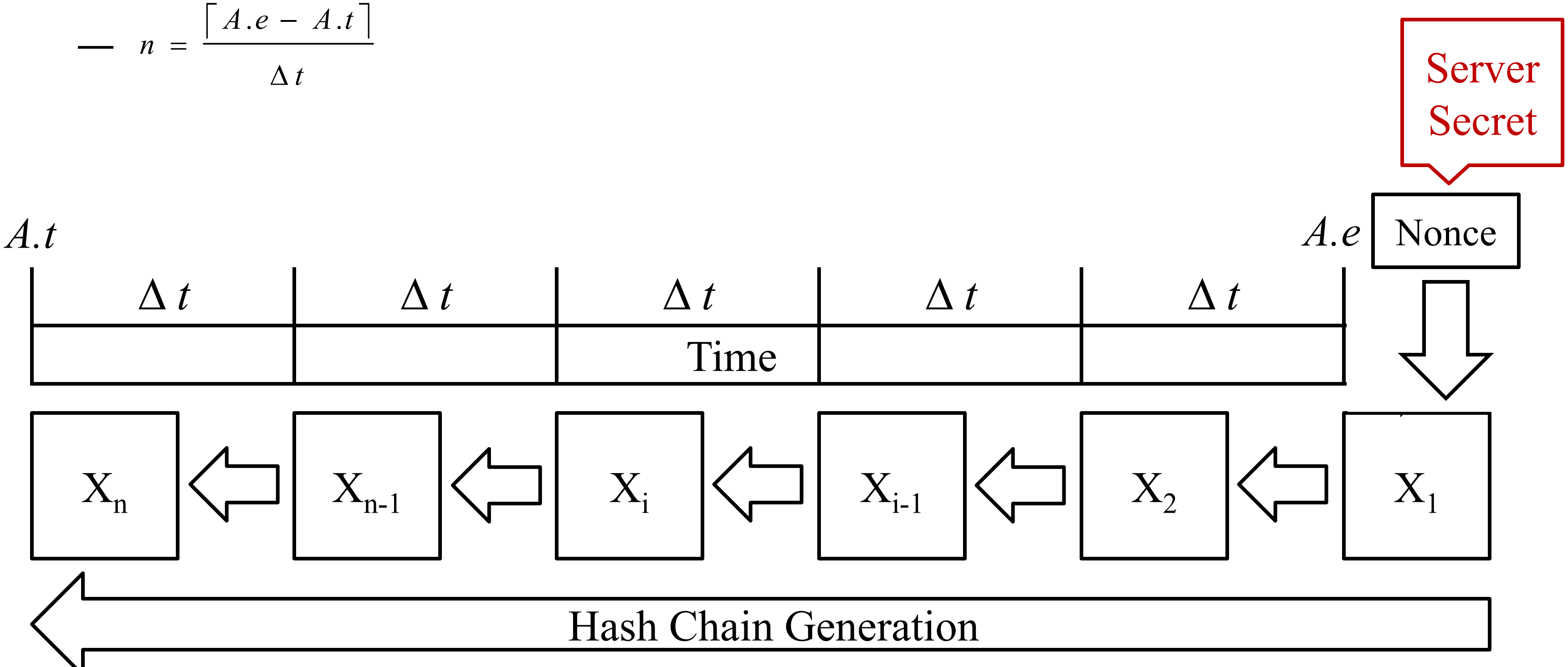
iHTTP: Sliding-Timestamps

- Based on the one way properties of hash chains
  - Authenticate X_i by hashing to known X_n
- Servers releases hash values to extend freshness



iHTTP: Hash Chain Generation

- Each hash operation represents time increment
  - Server defined and configurable: Δt
- Size of hash chain dependent on $A.t$, $A.e$, Δt
  - $n = \frac{\lceil A.e - A.t \rceil}{\Delta t}$

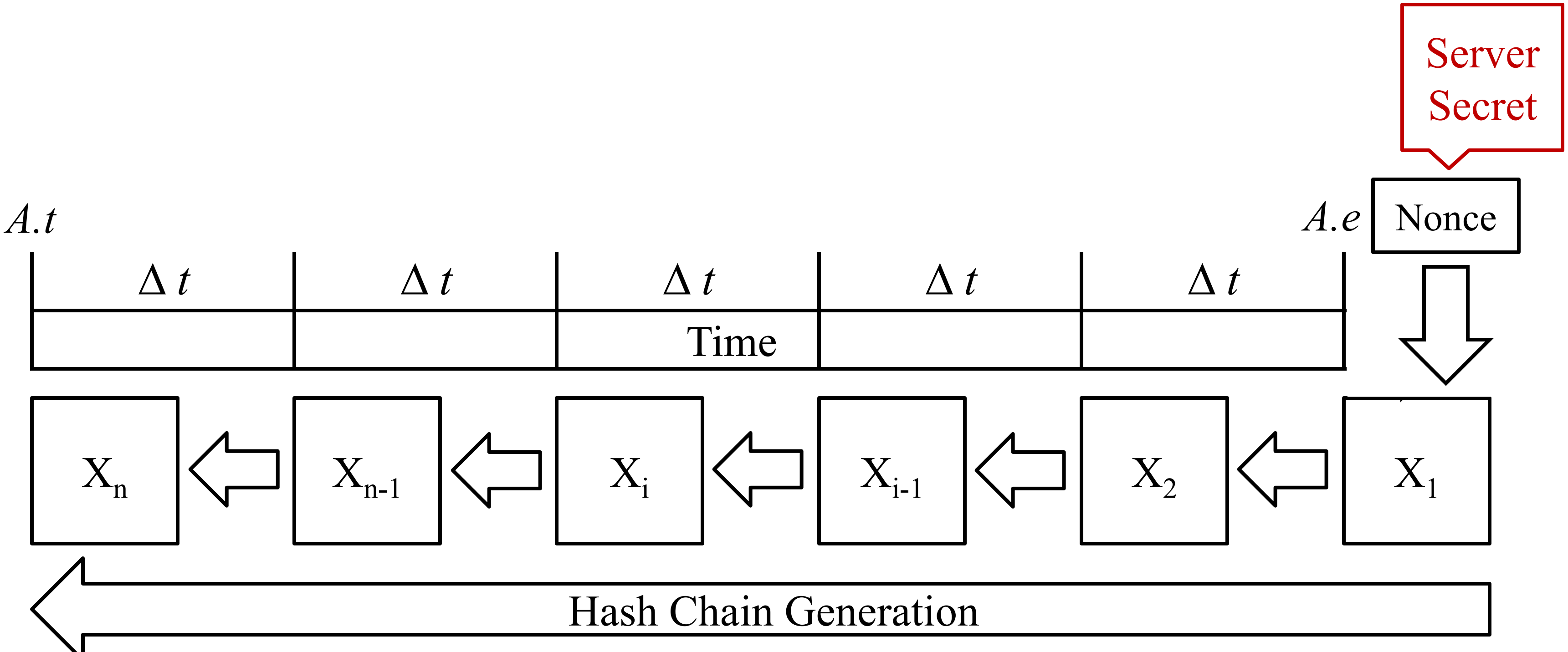


iHTTP: Hash Chain Generation cont.

- X_n and Δt are signed with the authenticator

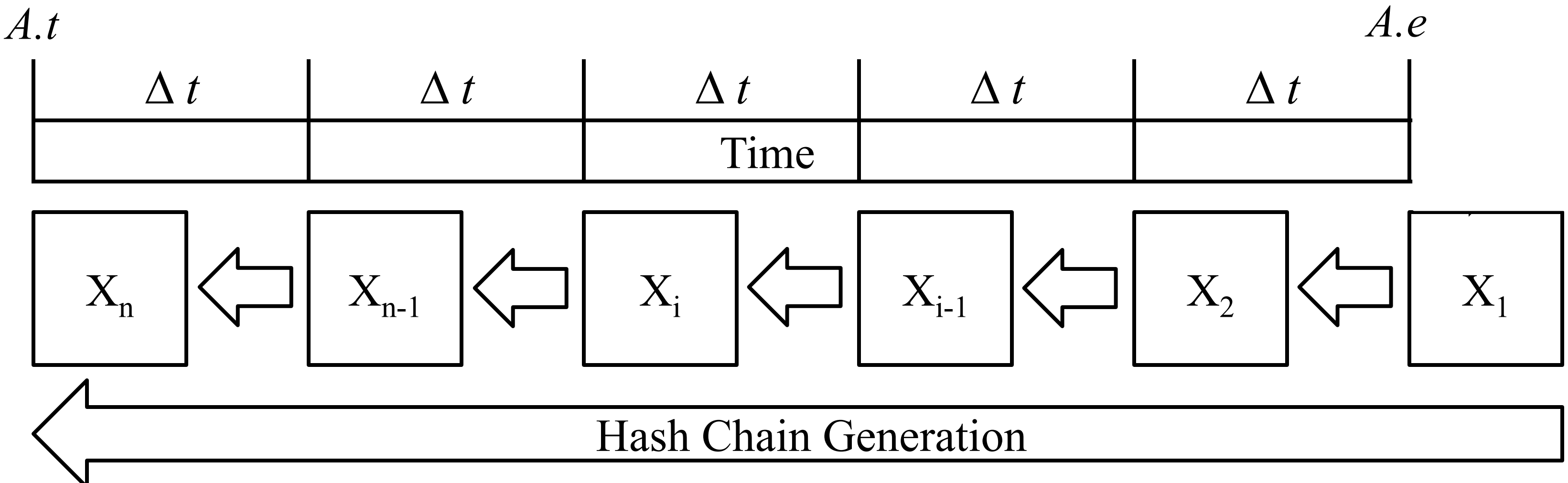
$$Sign_k\{H(A.t \mid A.e \mid A.u \mid A.l \mid HTTP.Headers \mid HTTP.Content \mid X_n \mid \Delta t)\}$$

- Server stores nonce in authenticator cache



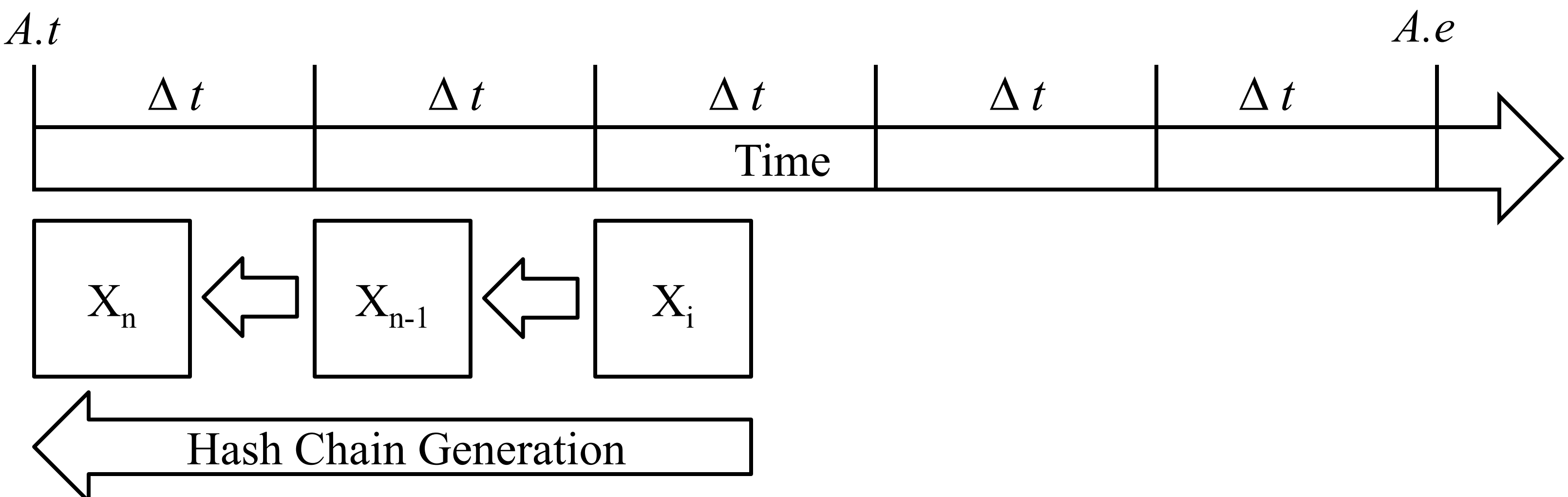
iHTTP: Extending a Timestamp

- Server releases X_i with each authenticator
  - Where i is based on current server time c
    - $i = n - \left\lceil \frac{c - A.t}{\Delta t} \right\rceil$
  - Sliding-Timestamp: $(n - i) * \Delta t + A.t$



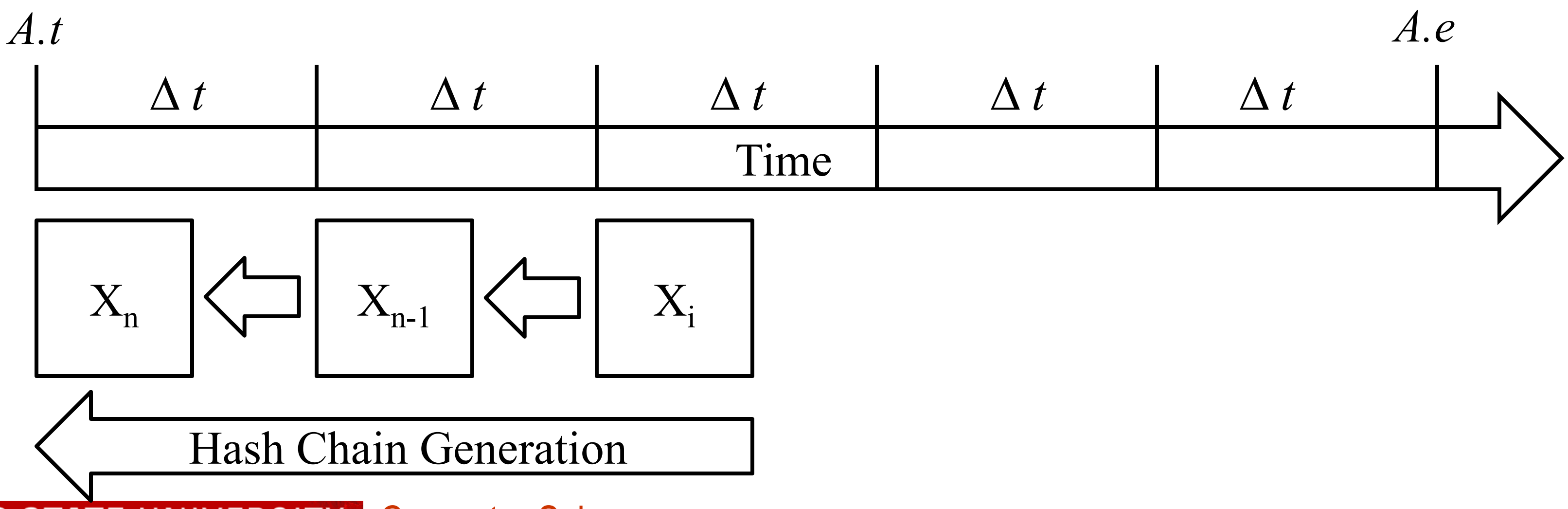
iHTTP: Client Authentication

- Verify the authenticator data using the signature
  - Verifies X_n and Δt
- Authenticate X_i by hashing X_i to X_n
  - $H^{n-i}(X_i) = X_n$



iHTTP: Client Authentication cont.

- Authenticate freshness
  - Sliding-Timestamp: $(n - i) * \Delta t + A.t$
    - Must be greater than request timestamp



iHTTP: Client Authenticator Verification

- Clients verify at least one signature per response
- Rendering web pages may require many signature verifications
  - CNN.com: 128 responses
  - All-Thats-Interesting.tumblr.com: 176 responses
- Clients can get overwhelmed
  - Especially for resource constrained clients

iHTTP: Opportunistic Hash Verification

- Client Request Anticipation
  - Servers parse HTML responses for resources
  - URLs are used to look up authenticators
  - Authenticators are hashed and embedded in HTML

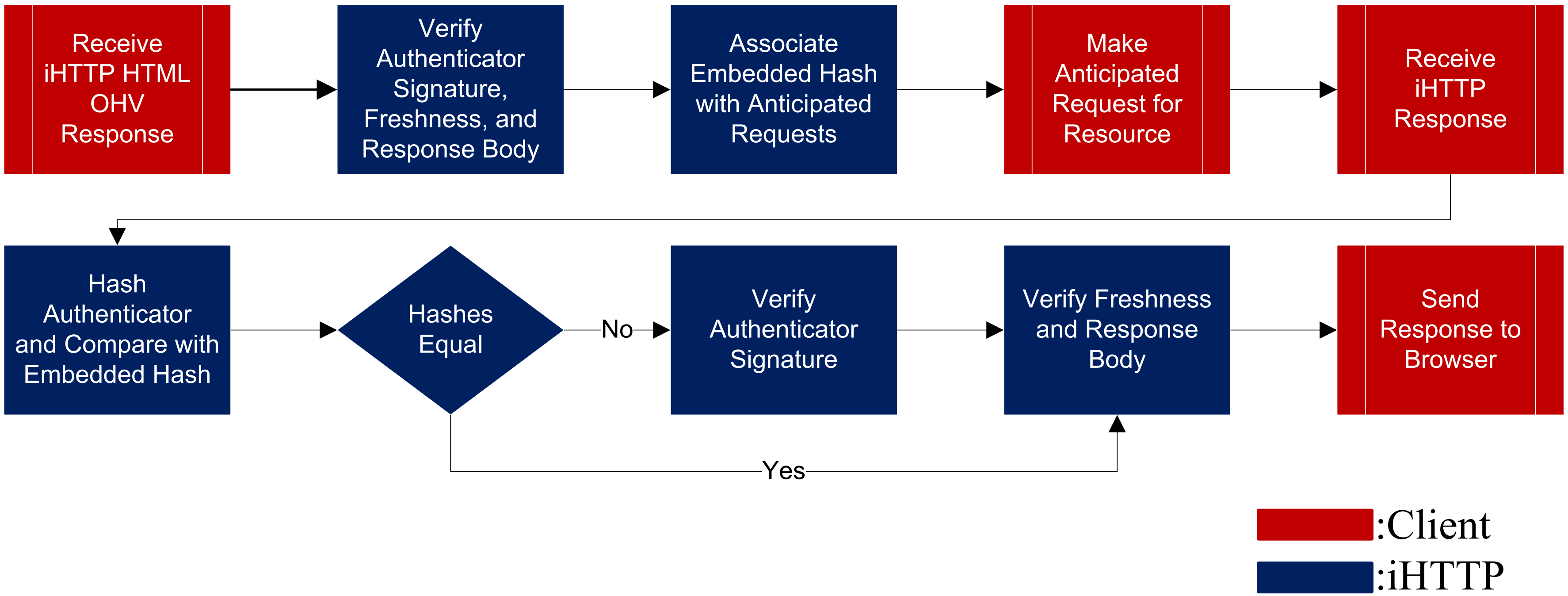
$$H(A.t \mid A.e \mid A.u \mid A.l \mid HTTP.Headers \mid HTTP.Content \mid X_n \mid \Delta t)$$

```
<html>
    <head>
        <link type="text/css" href="/screen_home.css"

    </head>
    …
</html>
```

iHTTP: Client Authenticator Verification

- Verifying HTML verifies embedded hashes
- Hashes can be used verify authenticators



Security Analysis

- Authenticated data
  - SBHI based on existing PKI infrastructure
  - Authenticators are bounds to data
- Data Integrity
  - Cryptographic hash representing data is signed
- Verifiable freshness
  - Clients calculate timestamps
    - Cannot forge due to one-way hash chain properties

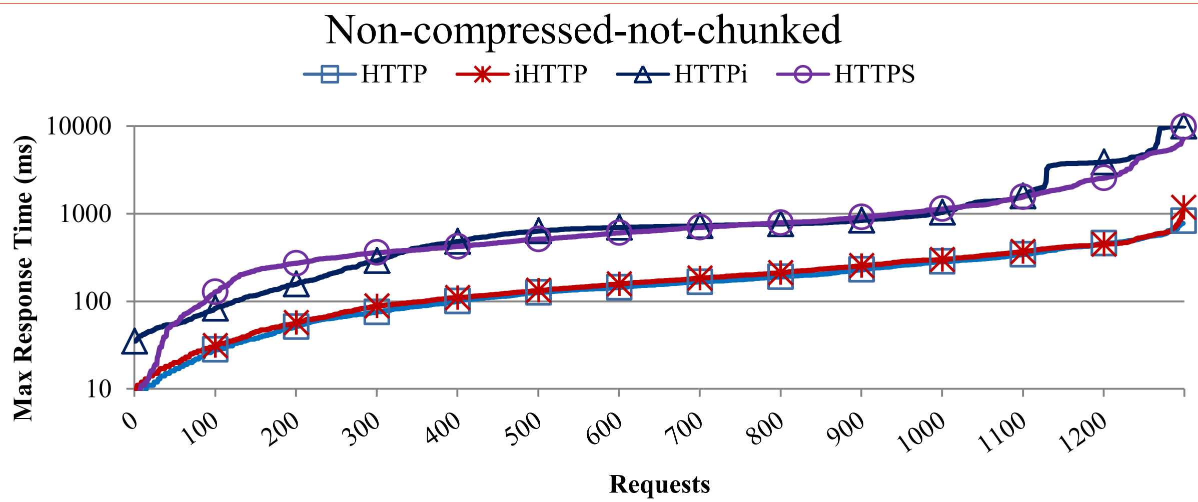
Limitation

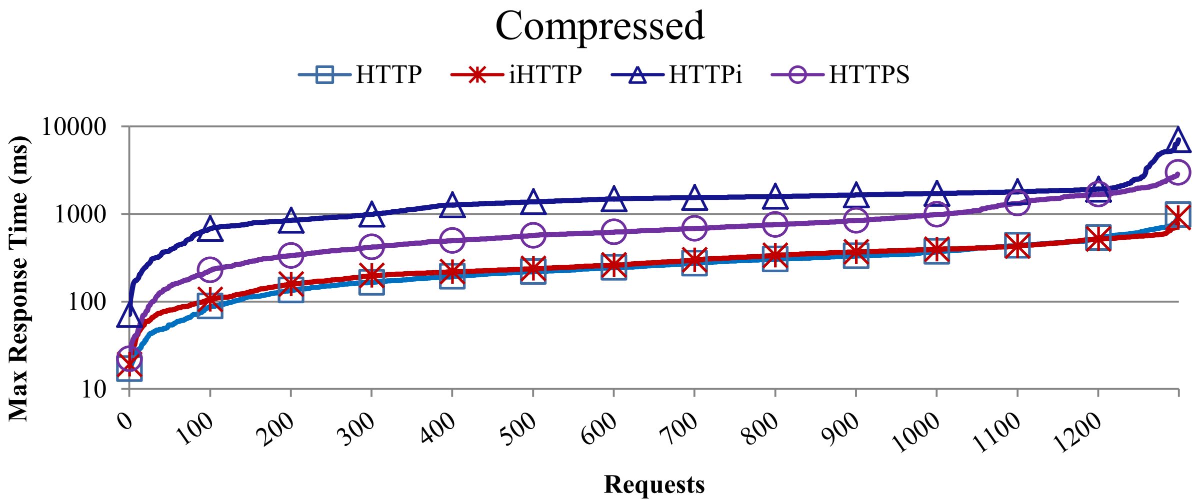
- SBHI not suited for all HTTP data
  - Very beneficial for Client-Static data
    - Client-Static data is cacheable
  - Not reliable for Client-Unique data
    - Attacks by providing logically incorrect data to users
    - Cannot protect client requests
      - Cookies and post data not protected
    - Each response must be signed
      - Poor performance

Experimental Evaluation

- Server Macrobenchmark
  - Compare iHTTP, HTTP, HTTPS, and HTTPi
  - Deployed blog template website
    - Client-Static data
  - Simulated 130 simultaneous clients
  - Four different configurations
    - Non-Chunked-Not-Compressed
    - Compressed
    - Chunked
    - Chunked-compressed

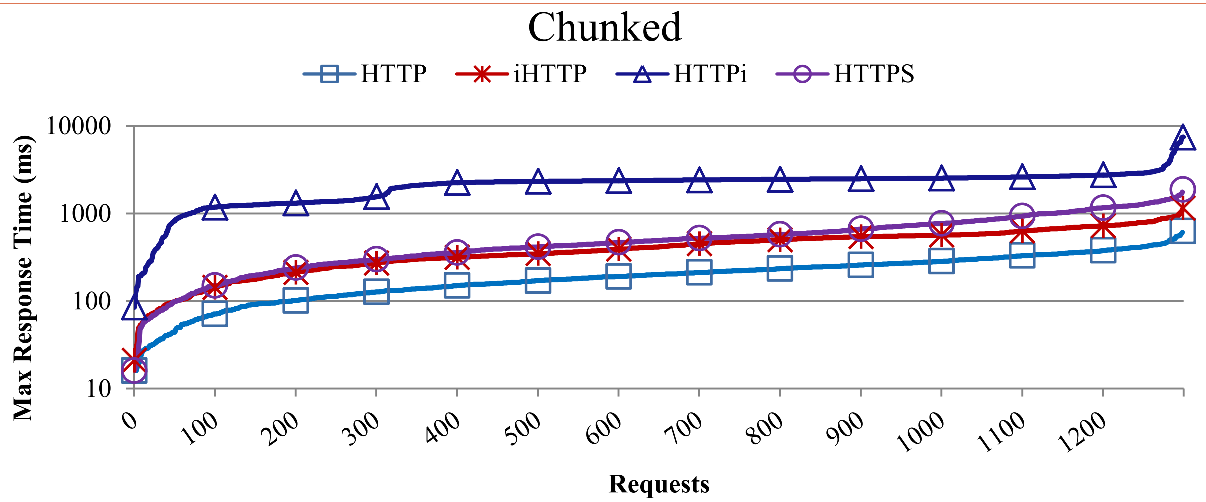
JMeter Benchmark Results

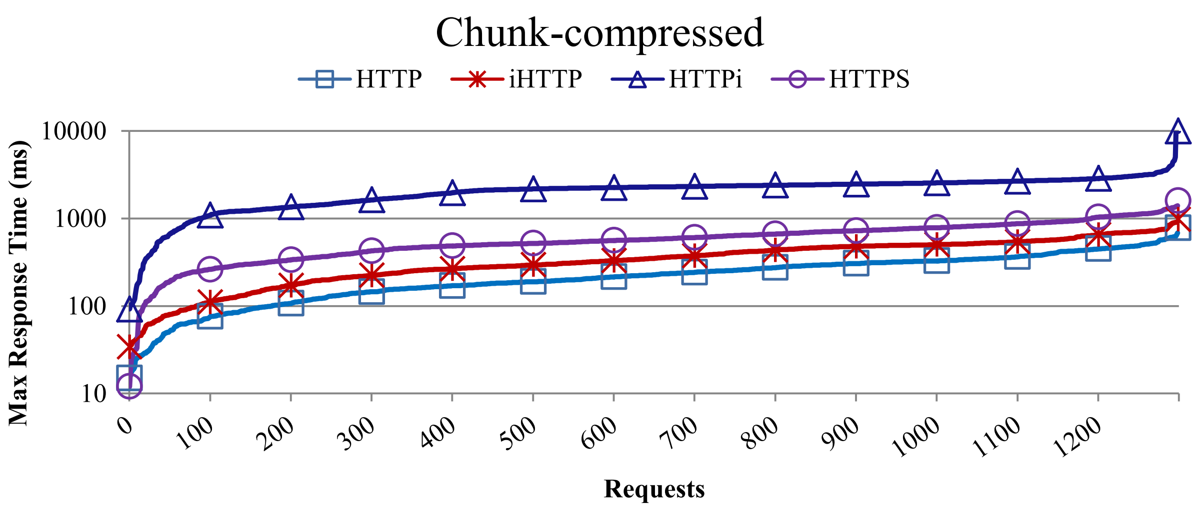




*X-Axis: Requests, Y-Axis: Max Response Time

JMeter Benchmark Results cont.





*X-Axis: Requests, Y-Axis: Max Response Time

Experimental Evaluation

- Client Macrobenchmark
  - Evaluate impact of Opportunistic Hash Verification*
    - On a resource constrained client
      - Android enabled Droid2
  - 20 requests to Client-Static blog
    - 17 resources per web page

Configuration	Avg Page Load
Non-OHV	7.4321 s
OHV	5.8291 s

* SBHI impact on clients evaluated in previous research

Conclusion

- iHTTP
  - Lightweight authentication of HTTP response data
  - Both secure and efficient
  - Enables freshness authentication without signing
    - Sliding-Timestamps
  - Minimizes client computation
    - Opportunistic Hash Verification
  - Performance
    - Better than previous SBHI approach HTTPi
    - Similar to HTTP for non-chunked Client-Static data

iHTTP: Efficient Authentication of Non-Confidential HTTP Traffic

Thank you

Questions?

Server Microbenchmark

- Five primary operations

Operation	**Time**
Authenticator Creation	4.97771 ms
Signature Generation	4.32070 ms
Hash Embedding	0.13189 ms
Cache Search	0.08751 ms
SHA-1 Operation	0.00042 ms

- Signature generation
  - 86% of authenticator generation cost

Server Macrobenchmark

- Investigate SBHI on Client-Unique data
- SpecWeb2009
  - Industry standard benchmark software
  - Deployed banking web application
    - 15 unique pages
    - Unique page responses for each user

Protocol	Avg Response Time	Avg Bytes per Req
HTTP	544 ms	41,818
HTTPS	576 ms	41,828
iHTTP	647 ms	50,627
HTTPi	662 ms	52,147