Append-only Authenticated Dictionaries (AADs)

Friday, December 7th, 2018
Modular Approach to Cloud Security (MACS) Project Meeting

Alin Tomescu, Vivek Bhupatiraju, Babis Papamanthou, Dimitris Papadopoulos, Nikos Triandopoulos, Srinivas Devadas

PKI: Not just an academic problem...

PKI: Not just an academic problem...

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Gmail account security in Iran

September 8, 2011

Posted by Eric Grosse, VP Security Engineering

We learned last week that the compromise of a Dutch company involved with verifying the authenticity of websites could have put the Internet communications of many Iranians at risk, including their Gmail. While Google's internal systems were not compromised, we are directly contacting possibly affected users and providing similar information below because our top priority is to protect the privacy and security of our users.

PKI: Not just an academic problem...

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Startups

Apps

Gadgets

Google Bans China's Website Certificate Authority After Security Breach

Catherine Shu @catherineshu / Apr 1, 2015

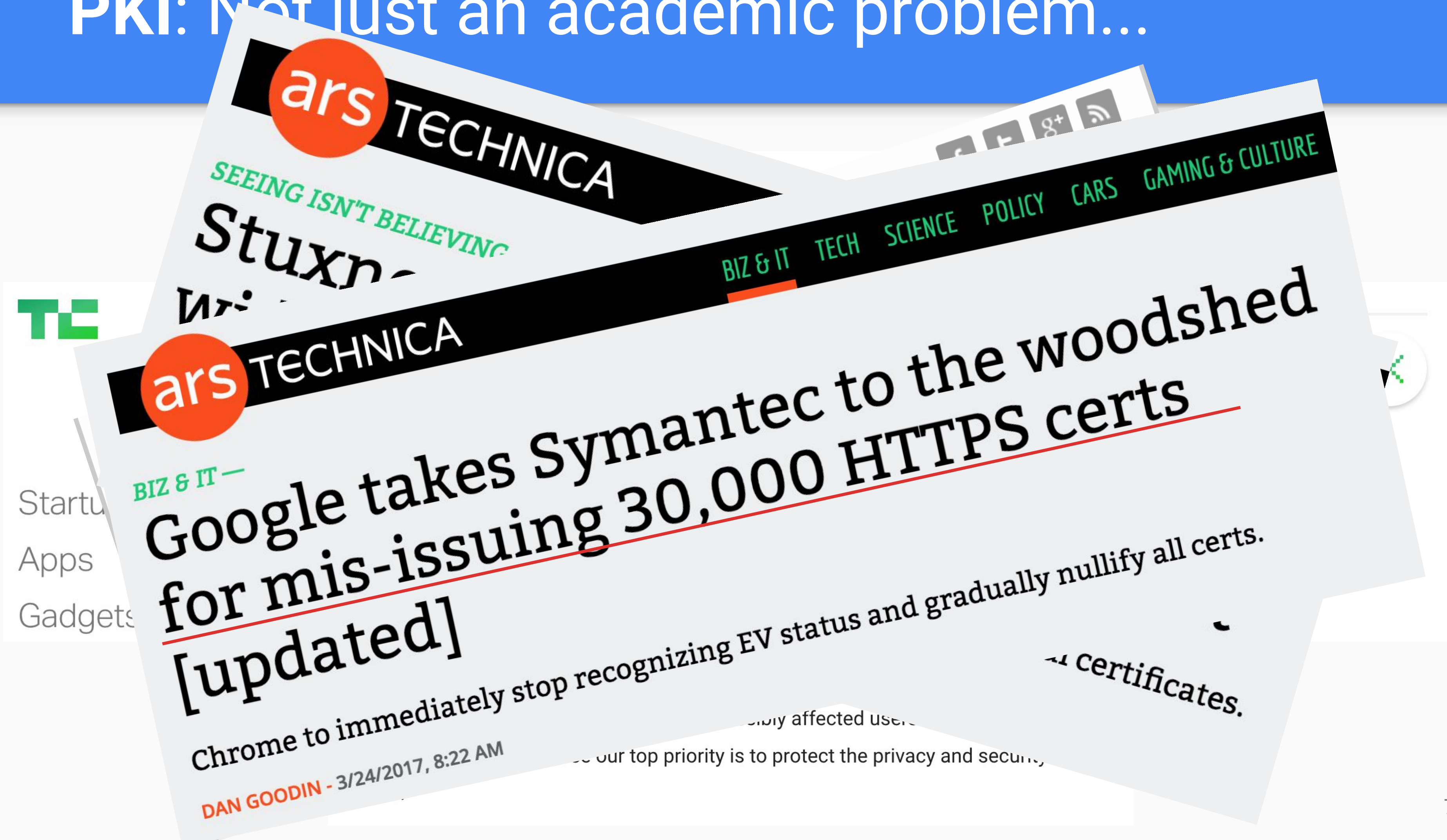
Comment

the authenticity of websites could have put the Internet communications of many Iranians at risk, including their Gmail. While Google's internal systems were not compromised, we are directly contacting possibly affected users and providing similar information below because our top priority is to protect the privacy and security of our users.

PKI: Not just an academic problem...

PKI: Not just an academic problem...

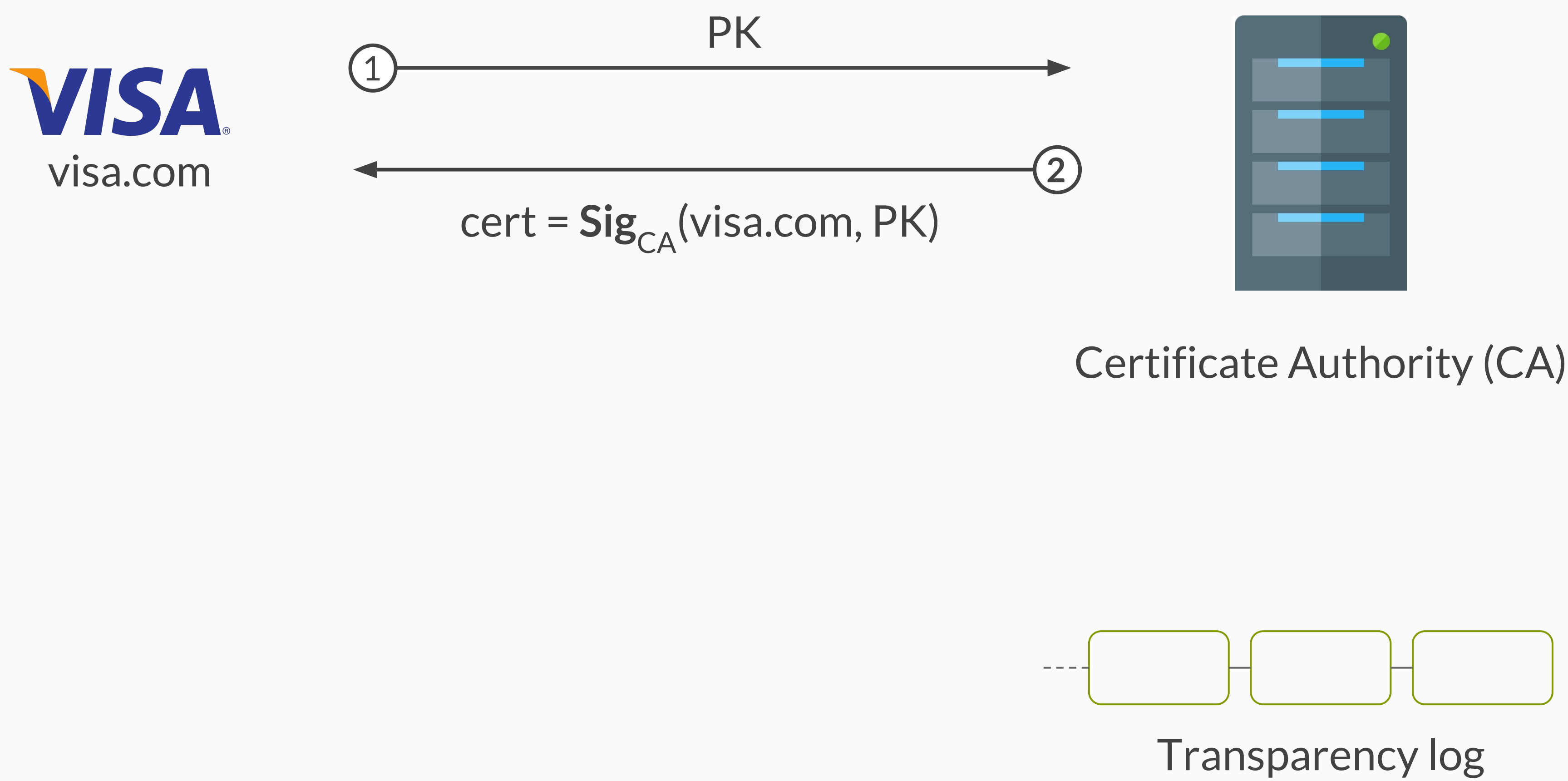
PKI: Not just an academic problem...



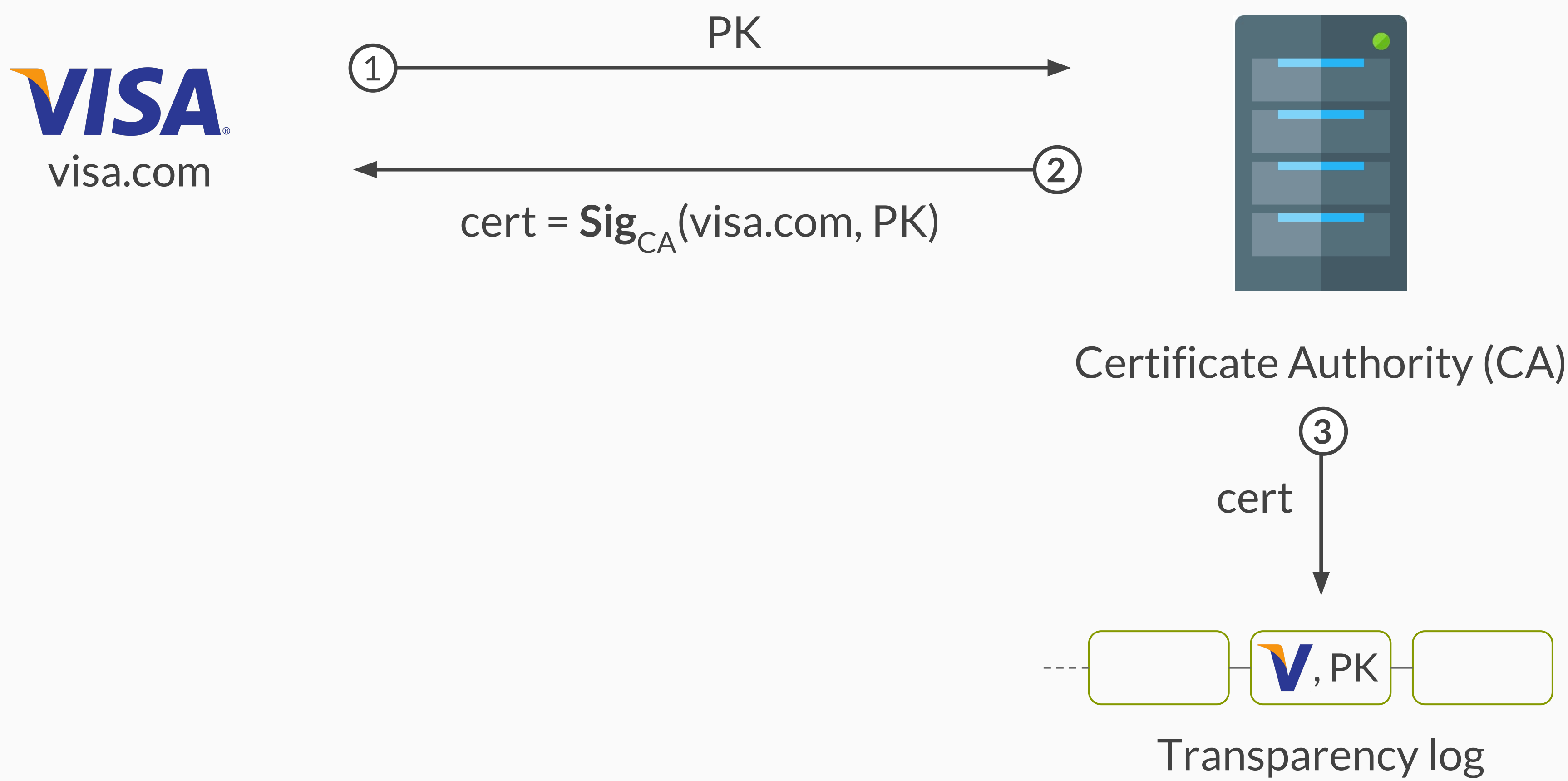
Certificate Transparency (CT) to the rescue

Certificate Transparency (CT) to the rescue

Certificate Transparency (CT) to the rescue



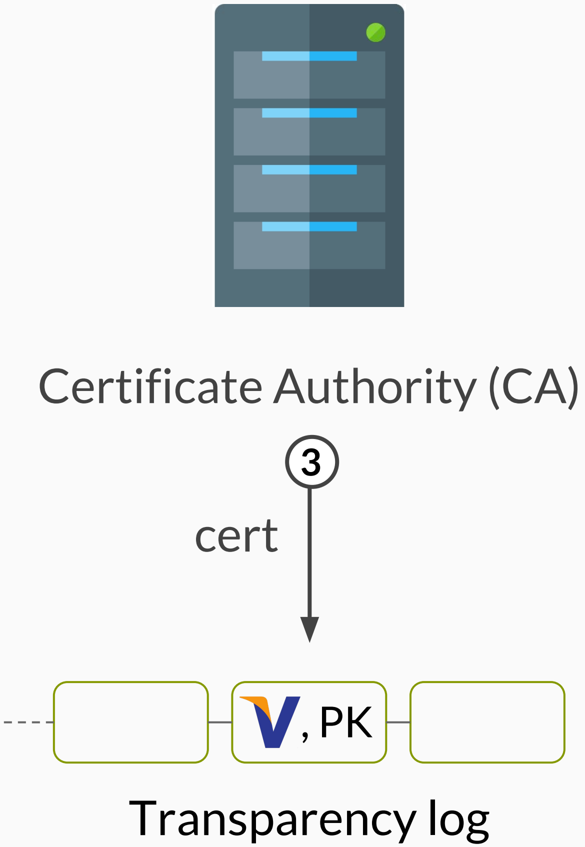
Certificate Transparency (CT) to the rescue



Certificate Transparency (CT) to the rescue

visa.com

Validity: A certificate is valid <u>only if</u> it's in the log.



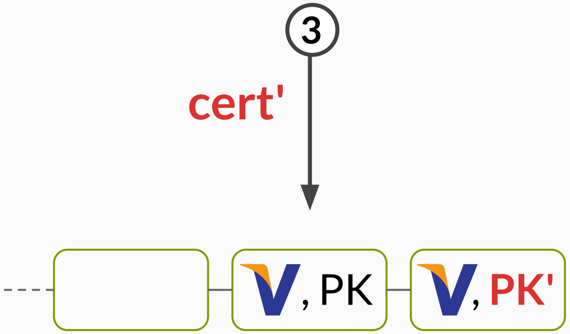
Certificate Transparency (CT) to the rescue

visa.com

Certificate Authority (CA)

Validity: A certificate is valid only if it's in the log.

Consequence: Fake certs must be published in the log.



Transparency log

Certificate Transparency (CT) to the rescue

visa.com

Certificate Authority (CA)

Transparency: Once certificate is in the log...

Transparency log

Certificate Transparency (CT) to the rescue

visa.com

Transparency: Once certificate is in the log, **(1)** it stays there forever and...

Certificate Authority (CA)

Transparency log

Certificate Transparency (CT) to the rescue

visa.com

Transparency: Once certificate is in the log, **(1)** it stays there forever and **(2)** it can be *efficiently* discovered.

Certificate Authority (CA)

Transparency log

Certificate Transparency (CT) to the rescue

visa.com

Certificate Authority (CA)

Transparency: Once certificate is in the log, **(1)** it stays there forever and **(2)** it can be *efficiently* discovered.

Non-equivocation: Everybody "sees" the same log.

Transparency log

Certificate Transparency (CT) to the rescue

visa.com

Certificate Authority (CA)

Transparency: Once certificate is in the log, **(1)** it stays there forever and **(2)** it can be ***efficiently*** discovered.

Non-equivocation: Everybody "sees" the same log.

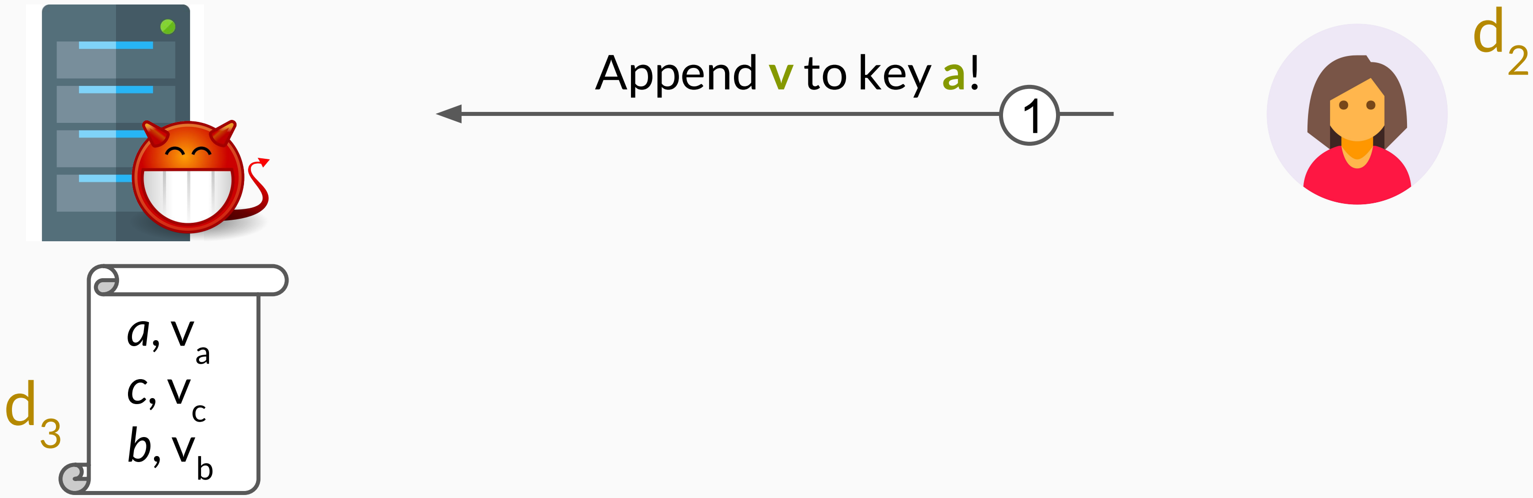
Consequence: Fake cert for VISA is discovered by VISA in the log.

Transparency log

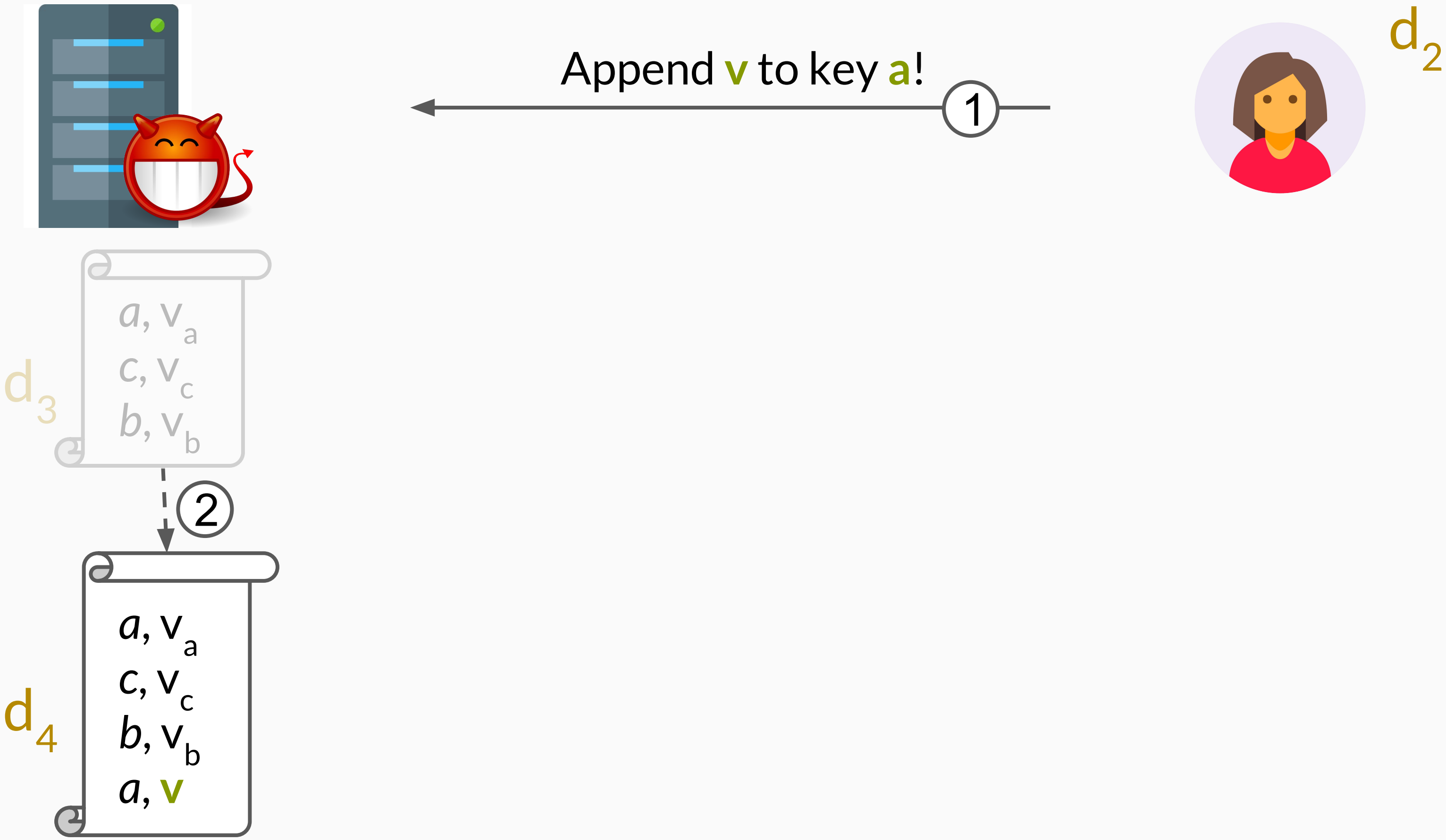
Certificate Transparency (CT) model

Certificate Transparency (CT) model

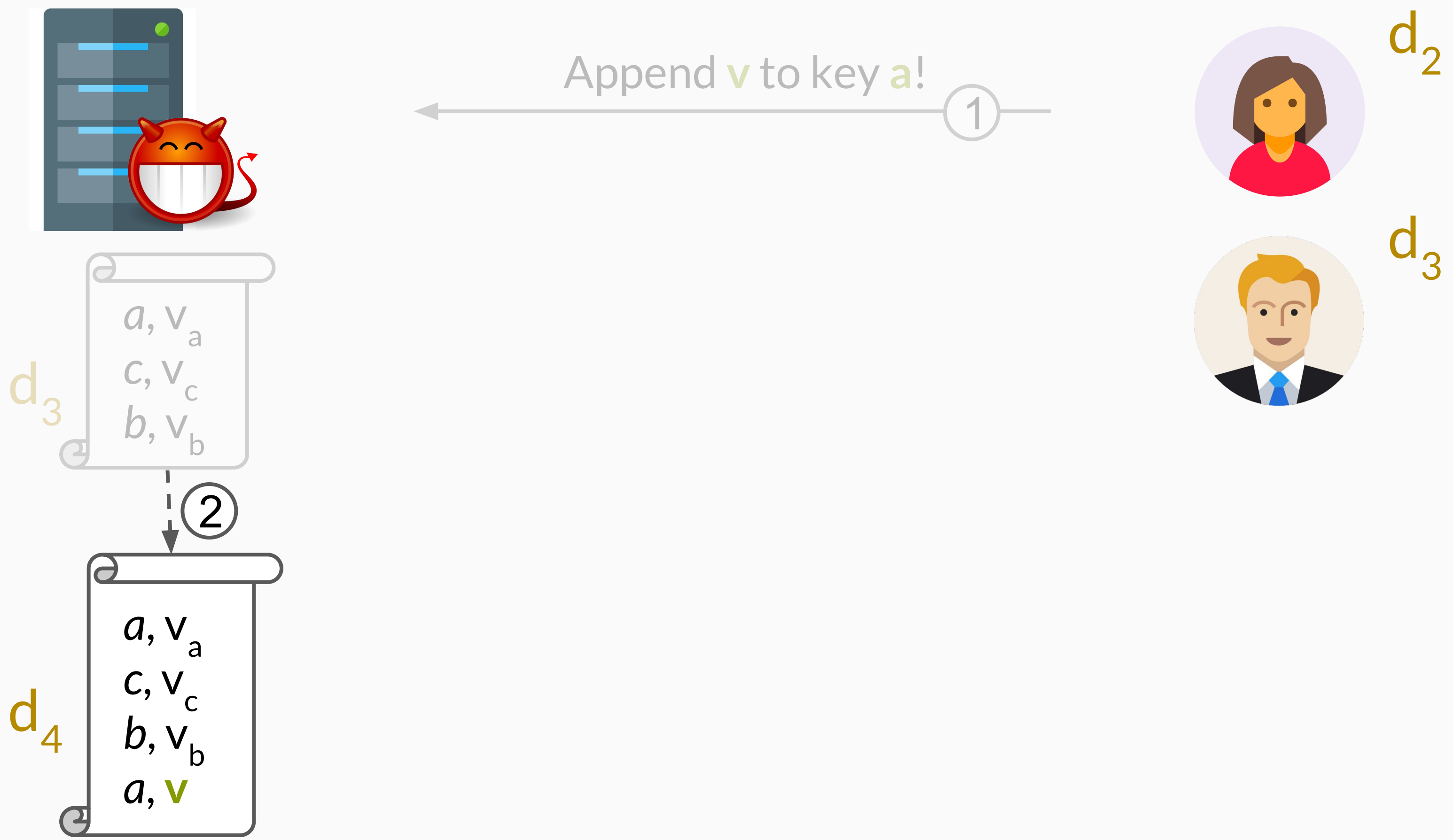
Certificate Transparency (CT) model



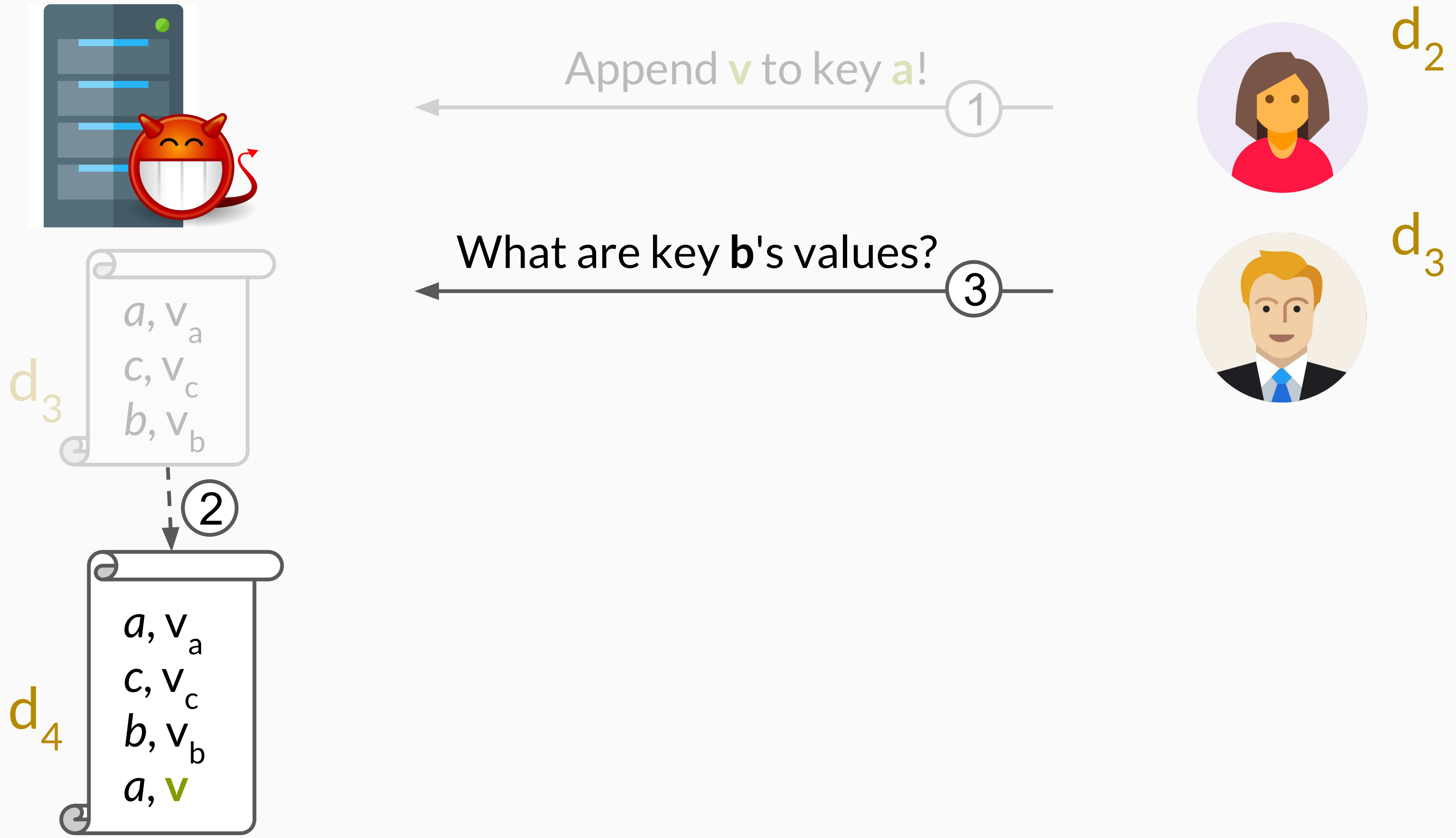
Certificate Transparency (CT) model



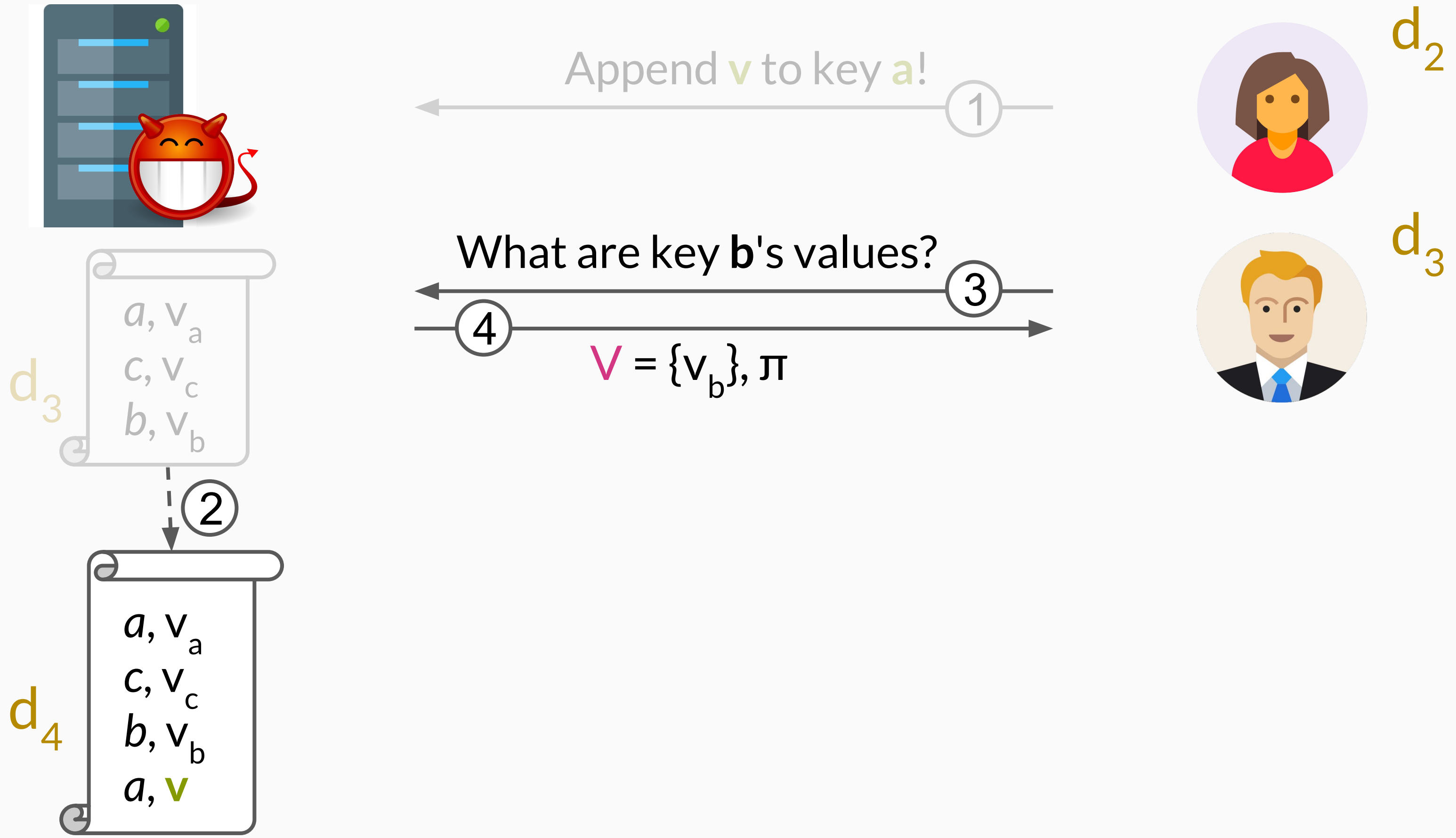
Certificate Transparency (CT) model



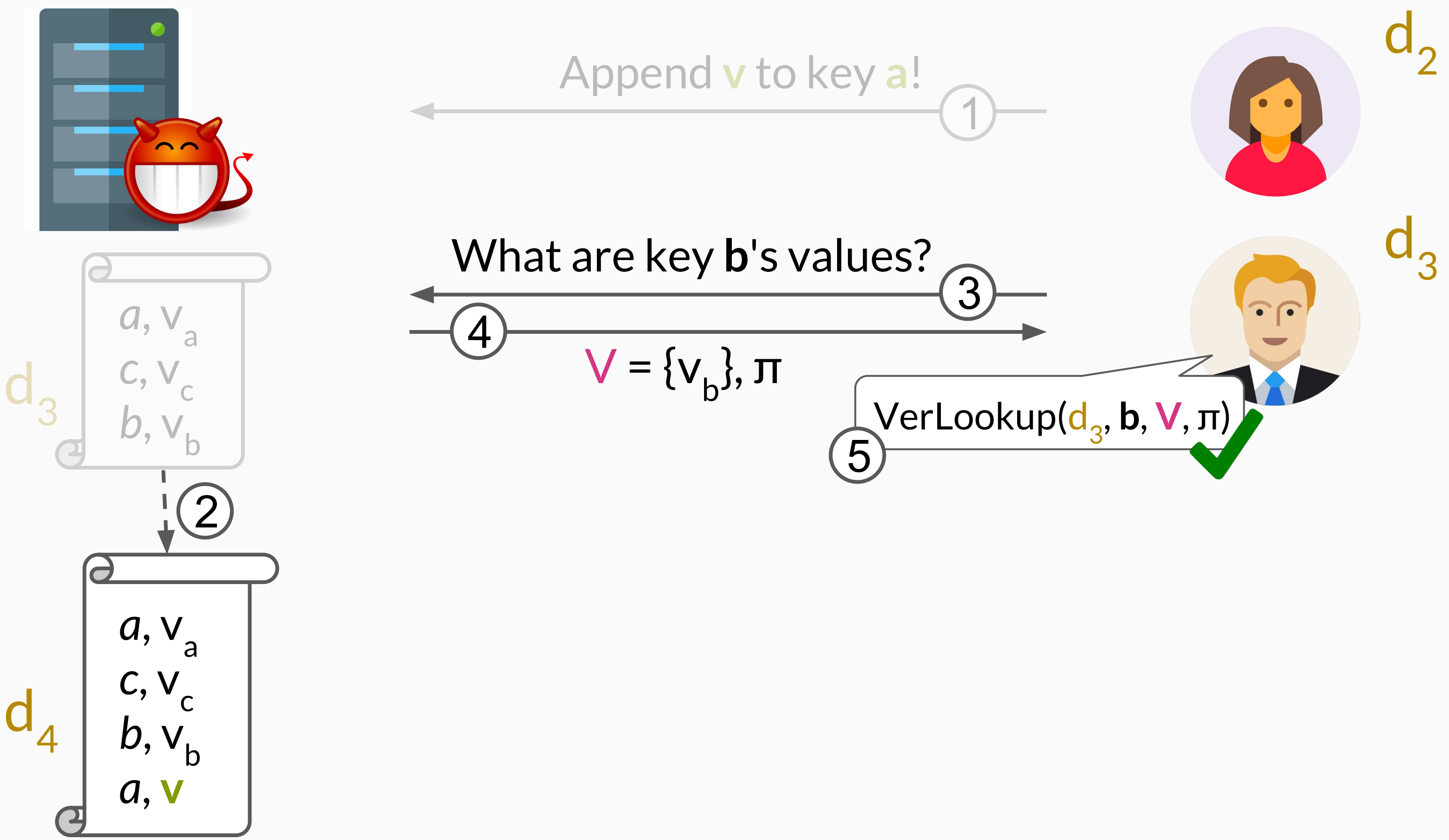
Certificate Transparency (CT) model



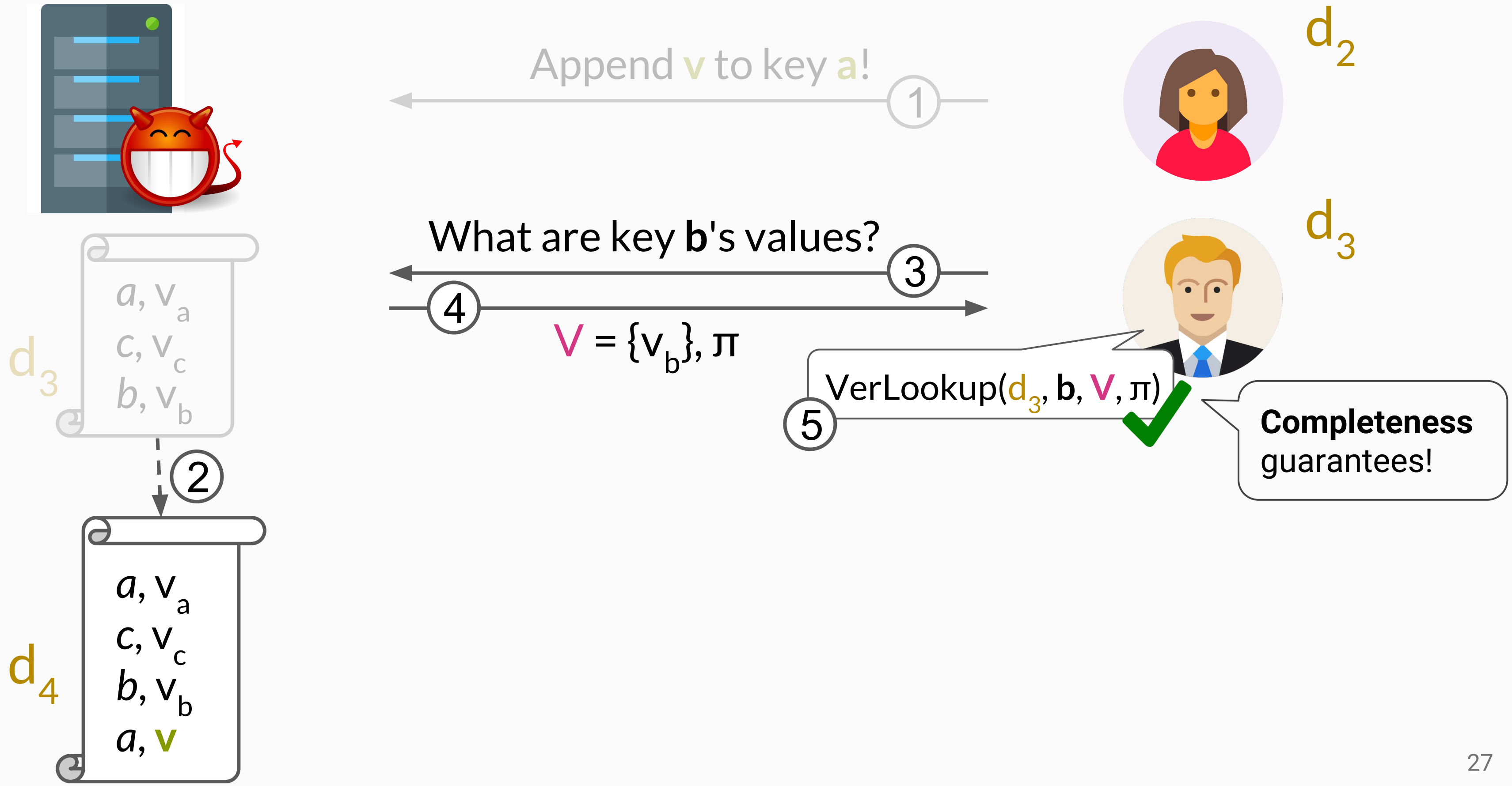
Certificate Transparency (CT) model



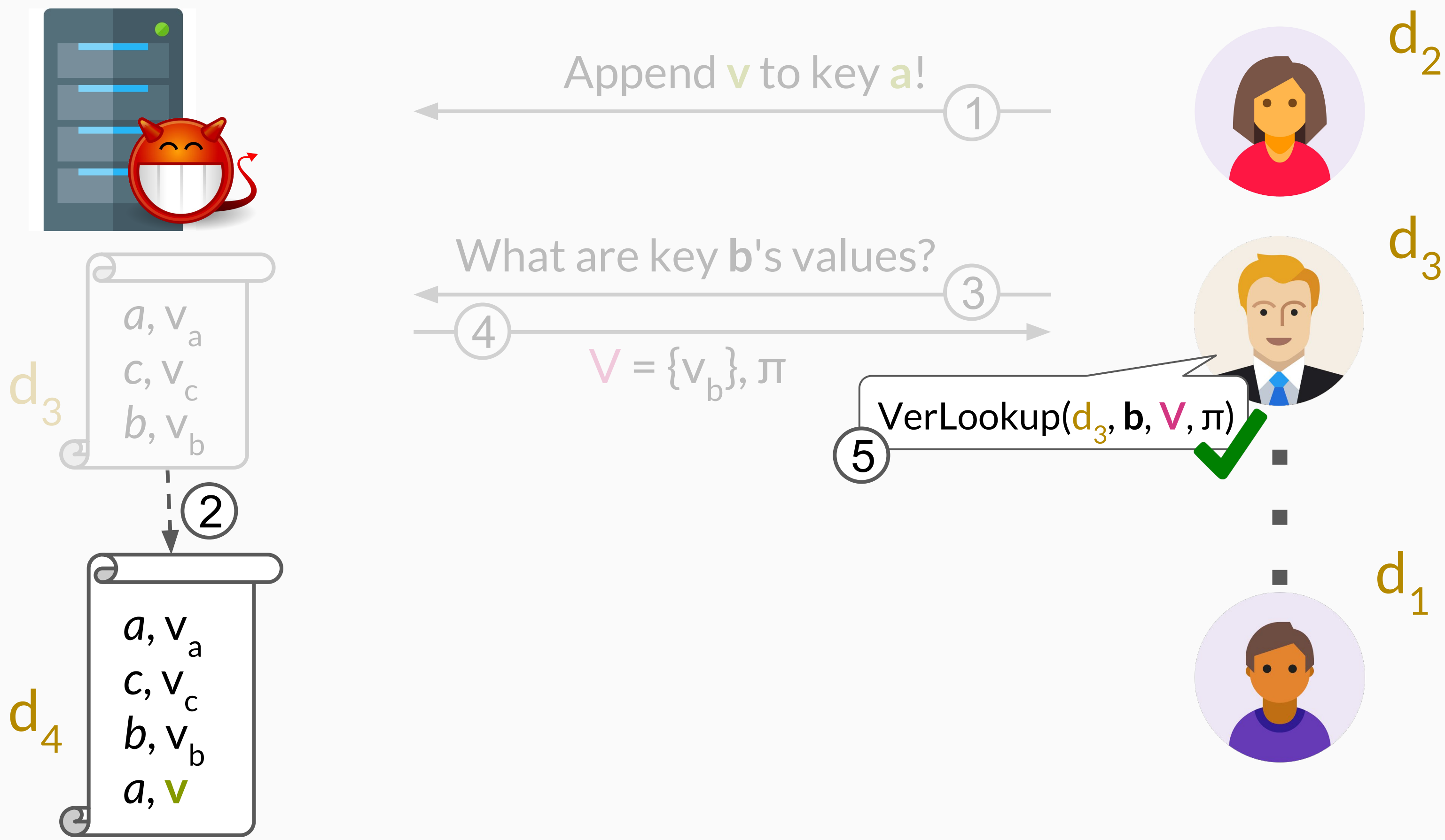
Certificate Transparency (CT) model



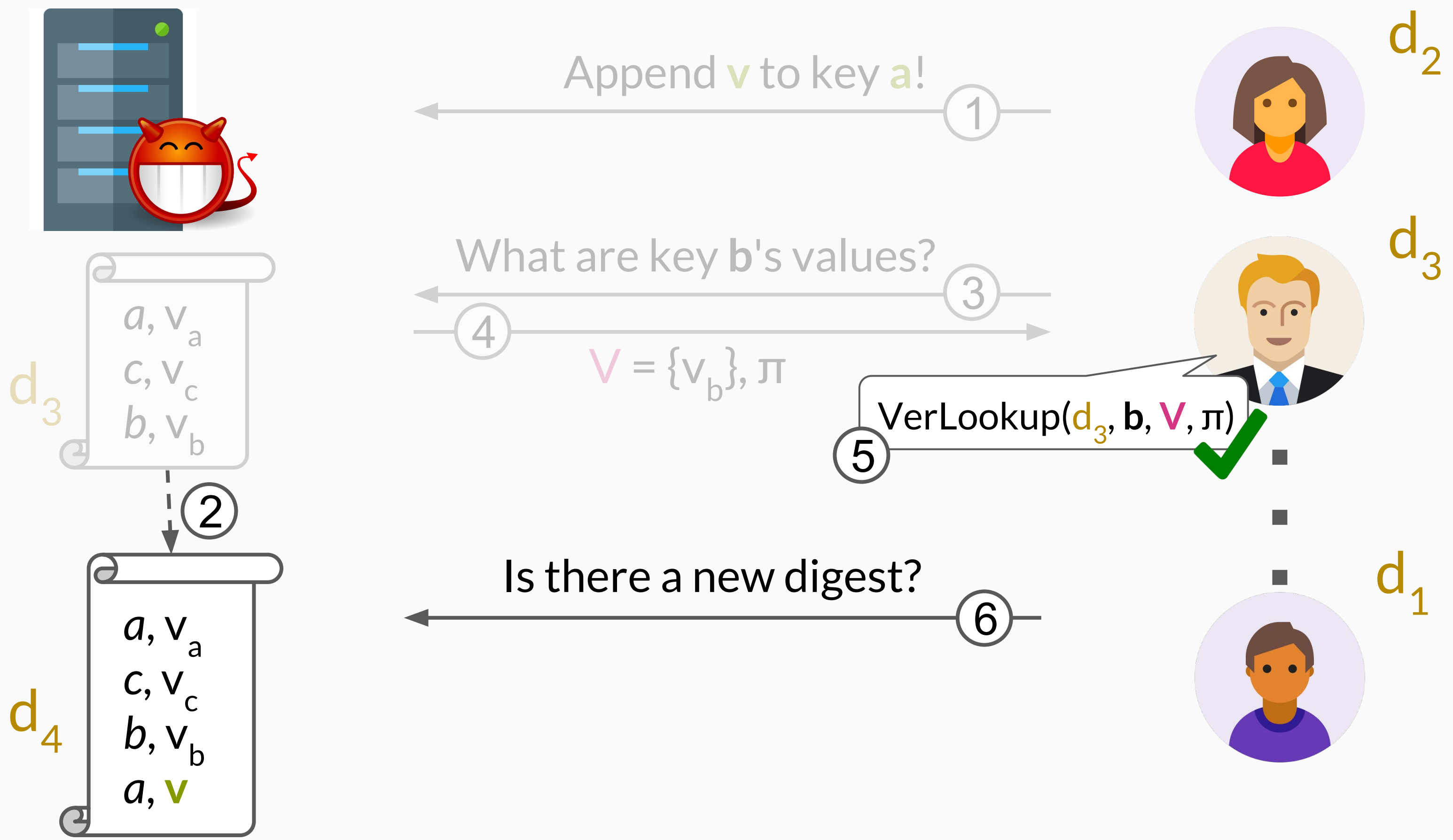
Certificate Transparency (CT) model



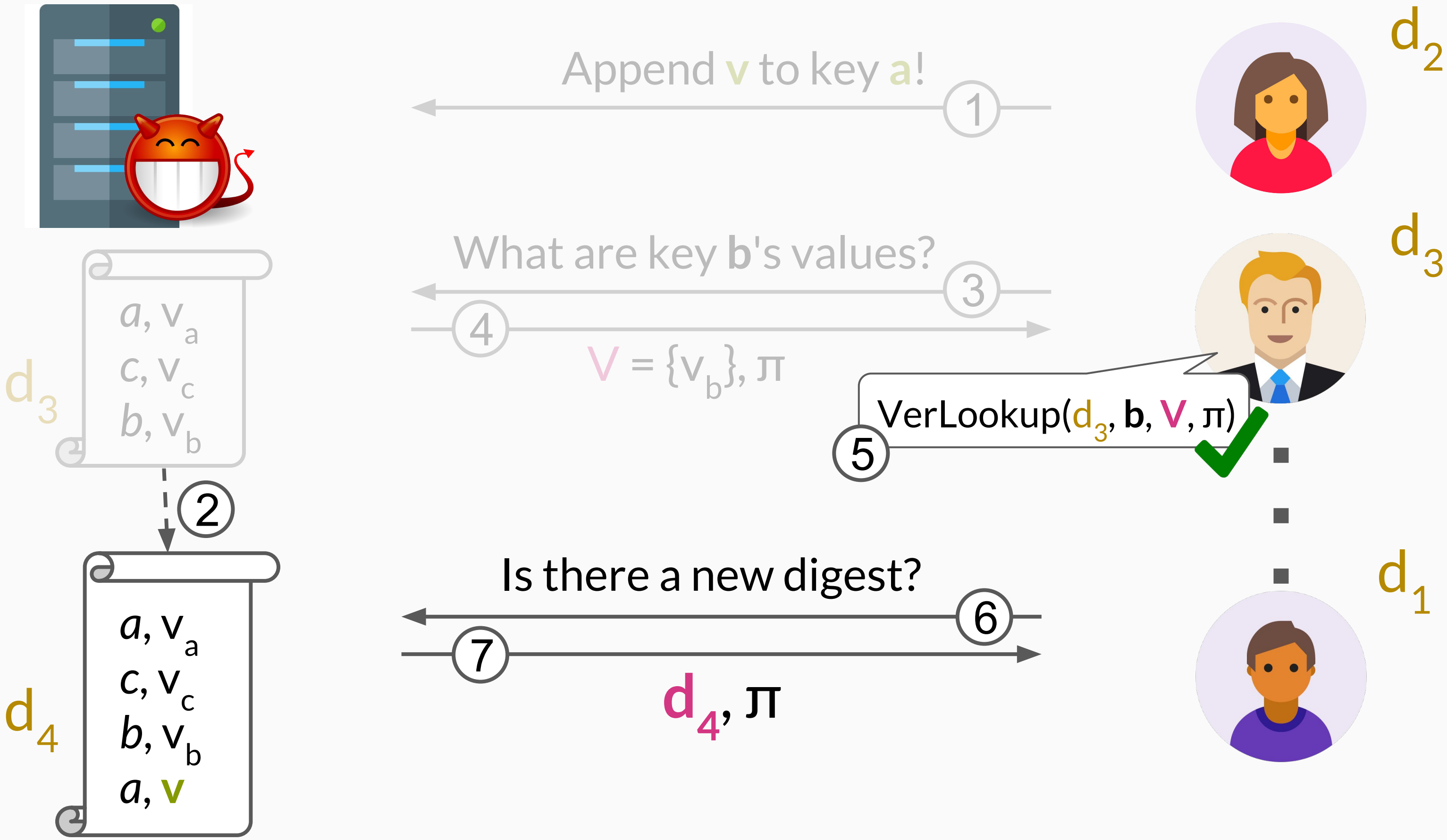
Certificate Transparency (CT) model



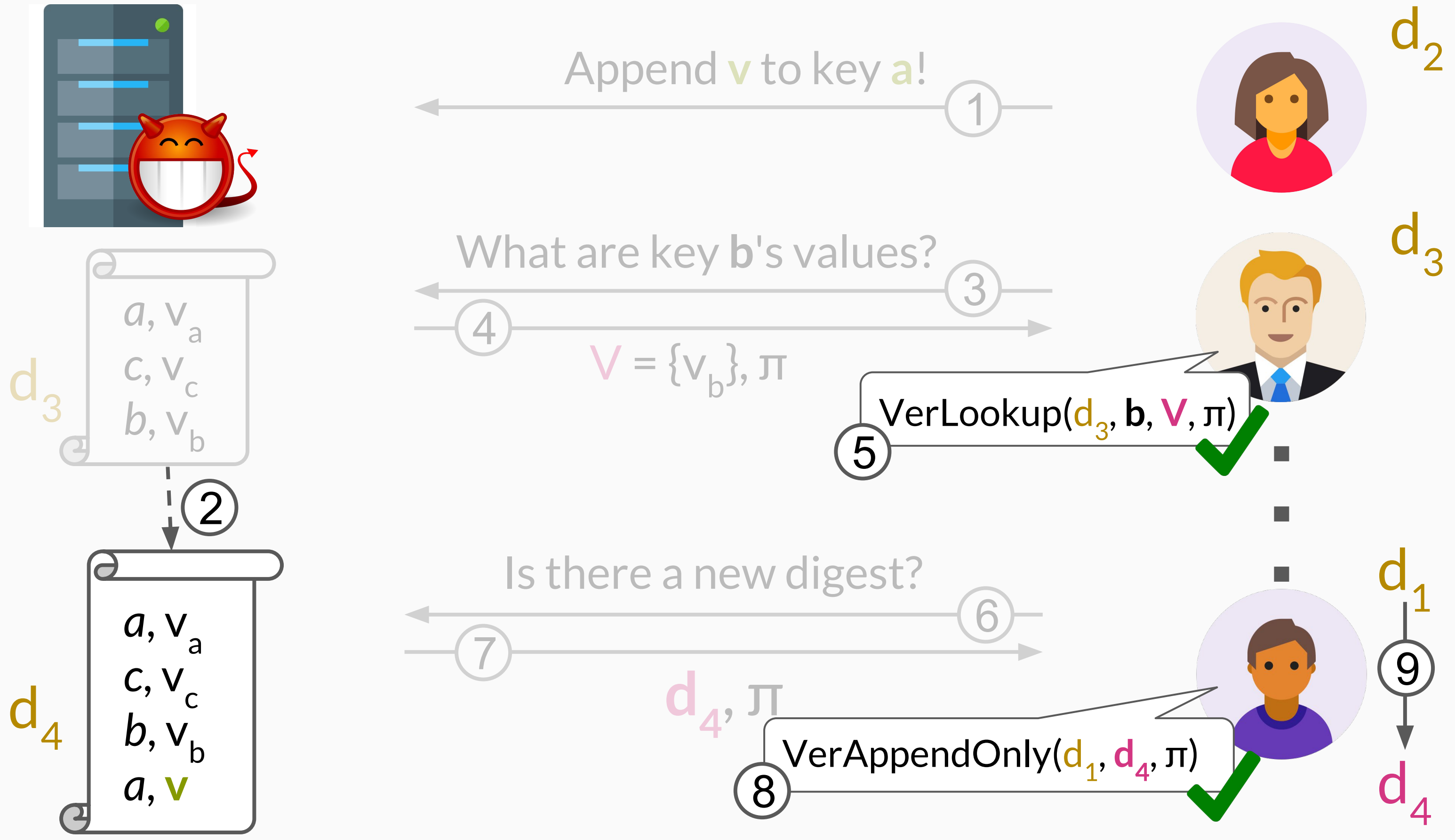
Certificate Transparency (CT) model



Certificate Transparency (CT) model



Certificate Transparency (CT) model



Previous work

Problem: In current logs, one of the proofs is large.

Previous work

Problem: In current logs, one of the proofs is large.

Transparency log	**Append** time	**Lookup proof** size	**Append-only proofs** size
CT	log n	n	log n

n = # of certificates in log

Previous work

Problem: In current logs, one of the proofs is large.

Transparency log	**Append** time	**Lookup proof** size	**Append-only proofs** size
CT	log n	n	log n
ECT, CONIKS, etc.	log n	log n	n

n = # of certificates in log

Our work:
Append-only Authenticated Dictionaries (AADs)

Problem: In current logs, one of the proofs is large.

Solution: AADs with polylogarithmic proof sizes!

reduce log bandwidth from **hundreds** of *GBps* down to **a few** *GBps!*

Transparency log	**Append** time	**Lookup proof** size	**Append-only proofs** size
CT	log n	n	log n
ECT, CONIKS, etc.	log n	log n	n
Our work	$\lambda \log^3 n$ (amortized)	$\log^2 n$	log n

n = # of certificates in log, λ = security parameter

Overview

In this talk: Append-only Authenticated Set (AAS) from bilinear accumulators

1. **Bilinear accumulators**
2. Bilinear Trees (BTs)
3. Bilinear Prefix Trees (BPTs)
4. Bilinear Frontier Trees (BFTs)
5. Amortization
6. *From AAS to AAD (not in this talk)*

Bilinear accumulators

Set **A** = {e_1, e_2, ..., e_n}, polynomial **α**(x) = (x - e_1)(x - e_2)...(x - e_n) with coefficients (a_0, a_1, ..., a_n)

q-SDH public parameters $\langle g, g^s, g^{s^2}, \ldots, g^{s^q} \rangle$, deg(**α**) < q. Commit to **α**(x) as follows:

$$\begin{aligned} \text{acc}(A) &= \left(g^{s^n}\right)^{a_n} \left(g^{s^{n-1}}\right)^{a_{n-1}} \ldots (g^s)^{a_1} (g)^{a_0} \\ &= g^{a_n s^n} g^{a_{n-1} s^{n-1}} \ldots g^{a_1 s} g^{a_0} \\ &= g^{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0} \\ &= g^{\alpha(s)} \end{aligned}$$

The commitment **acc(A)** is a *bilinear accumulator*. **Expensive:** O(n $\log^2$ n) time

Accumulator subset proofs

Let **A** with polynomial **α**(x), accumulator **a**,
and let **B** with polynomial **β**(x), accumulator **b,**

Accumulator subset proofs

Let **A** with polynomial **α**(x), accumulator **a**,
and let **B** with polynomial **β**(x), accumulator **b,**

$$A \subseteq B$$

Accumulator subset proofs

Let **A** with polynomial **α**(x), accumulator **a**,
and let **B** with polynomial **β**(x), accumulator **b,**

$$A \subseteq B \Leftrightarrow \alpha(x) \mid \beta(x) \Leftrightarrow \beta(x) = q(x)\alpha(x)$$

Accumulator subset proofs

Let **A** with polynomial **α**(x), accumulator **a**,
and let **B** with polynomial **β**(x), accumulator **b,**

$$A \subseteq B \Leftrightarrow \alpha(x) \mid \beta(x) \Leftrightarrow \beta(x) = q(x)\alpha(x)$$

Subset proof is $g^{q(s)}$ and is verified using bilinear map **e**():

Accumulator subset proofs

Let **A** with polynomial **α**(x), accumulator **a**,
and let **B** with polynomial **β**(x), accumulator **b,**

$$A \subseteq B \Leftrightarrow \alpha(x) \mid \beta(x) \Leftrightarrow \beta(x) = q(x)\alpha(x)$$

Subset proof is $g^{q(s)}$ and is verified using bilinear map **e**():

$$e(a, g^{q(s)}) = e(b, g) \Leftrightarrow$$

Accumulator subset proofs

Let **A** with polynomial **α**(x), accumulator **a**,
and let **B** with polynomial **β**(x), accumulator **b,**

$$A \subseteq B \Leftrightarrow \alpha(x) \mid \beta(x) \Leftrightarrow \beta(x) = q(x)\alpha(x)$$

Subset proof is $g^{q(s)}$ and is verified using bilinear map **e**():

$$e(a, g^{q(s)}) = e(b, g) \Leftrightarrow$$
$$e(g, g)^{\alpha(s)q(s)} = e(g, g)^{\beta(s)} \overset{*}{\Leftrightarrow}$$

Accumulator subset proofs

Let **A** with polynomial **α**(x), accumulator **a**,
and let **B** with polynomial **β**(x), accumulator **b,**

$$A \subseteq B \Leftrightarrow \alpha(x) \mid \beta(x) \Leftrightarrow \beta(x) = q(x)\alpha(x)$$

Subset proof is $g^{q(s)}$ and is verified using bilinear map **e**():

$$e(a, g^{q(s)}) = e(b, g) \Leftrightarrow$$
$$e(g, g)^{\alpha(s)q(s)} = e(g, g)^{\beta(s)} \overset{*}{\Leftrightarrow}$$
$$\beta(x) = q(x)\alpha(x)$$

(under q-SBDH)*

Accumulator subset proofs

Let **A** with polynomial **α**(x), accumulator **a**,
and let **B** with polynomial **β**(x), accumulator **b,**

$$A \subseteq B \Leftrightarrow \alpha(x) \mid \beta(x) \Leftrightarrow \beta(x) = q(x)\alpha(x)$$

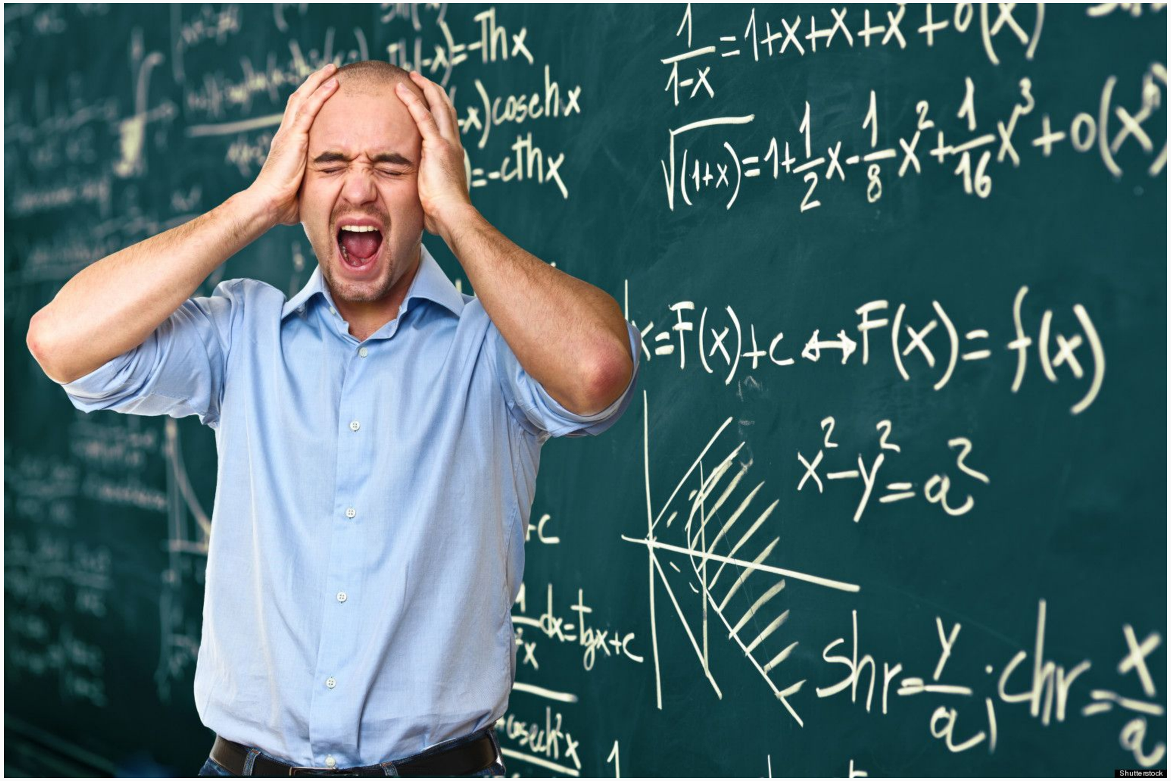
Subset proof is $g^{\mathbf{q(s)}}$ and is verified using bilinear map **e**():

$$e(a, g^{q(s)}) = e(b, g) \Leftrightarrow$$
$$e(g, g)^{\alpha(s)q(s)} = e(g, g)^{\beta(s)} \overset{*}{\Leftrightarrow}$$
$$\beta(x) = q(x)\alpha(x)$$

Expensive: O(n log n) time to compute **one** proof

(under q-SBDH)*

Accumulator disjointness proofs



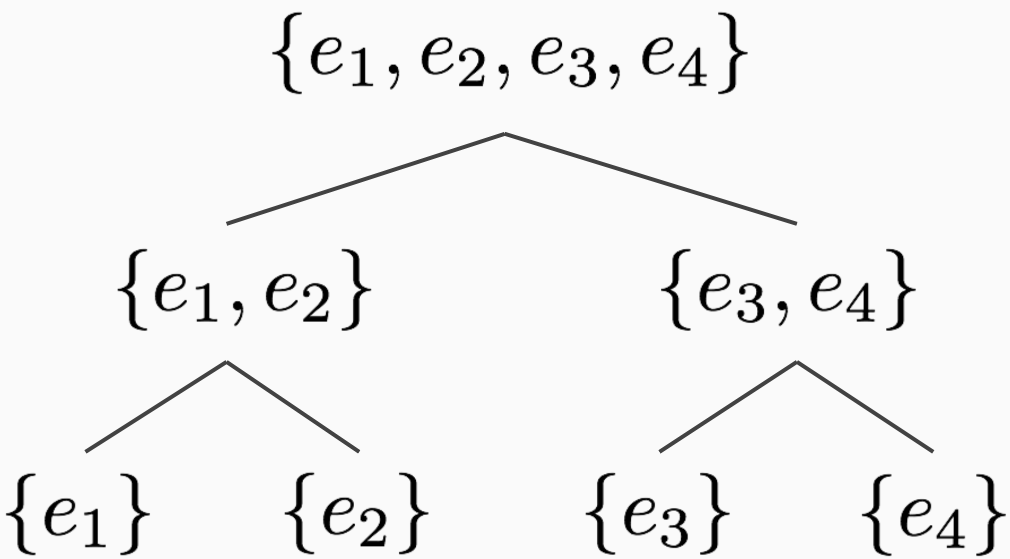
The road so far...

1. Bilinear accumulators
2. **Bilinear Trees (BTs)**
3. Bilinear Prefix Trees (BPTs)
4. Bilinear Frontier Trees (BFTs)
5. Amortization
6. *From AAS to AAD (not in this talk)*

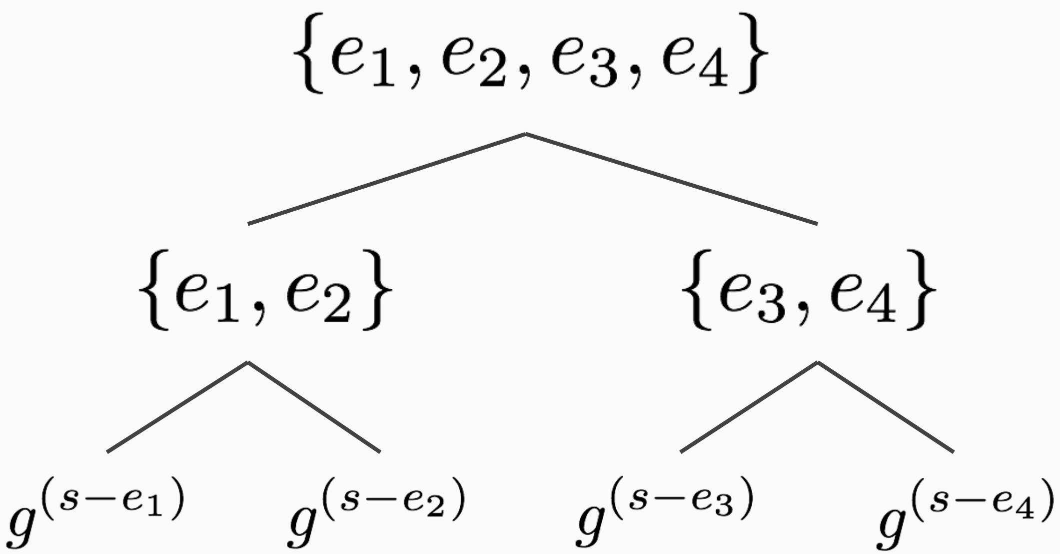
Bilinear Trees (BTs):
Precomputed membership proofs

$$\{e_1, e_2, e_3, e_4\}$$

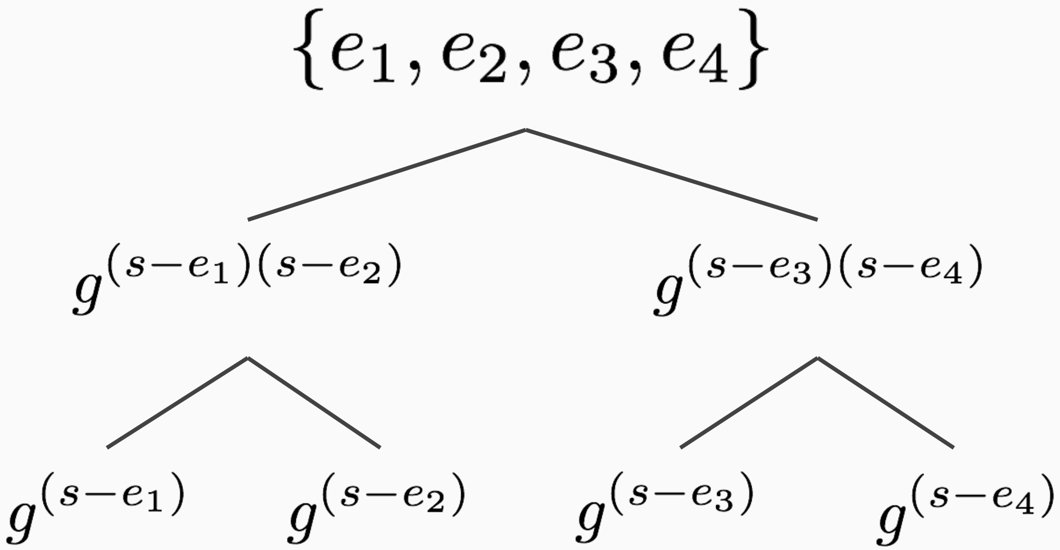
Bilinear Trees (BTs):
Precomputed membership proofs



Bilinear Trees (BTs):
Precomputed membership proofs



Bilinear Trees (BTs):
Precomputed membership proofs



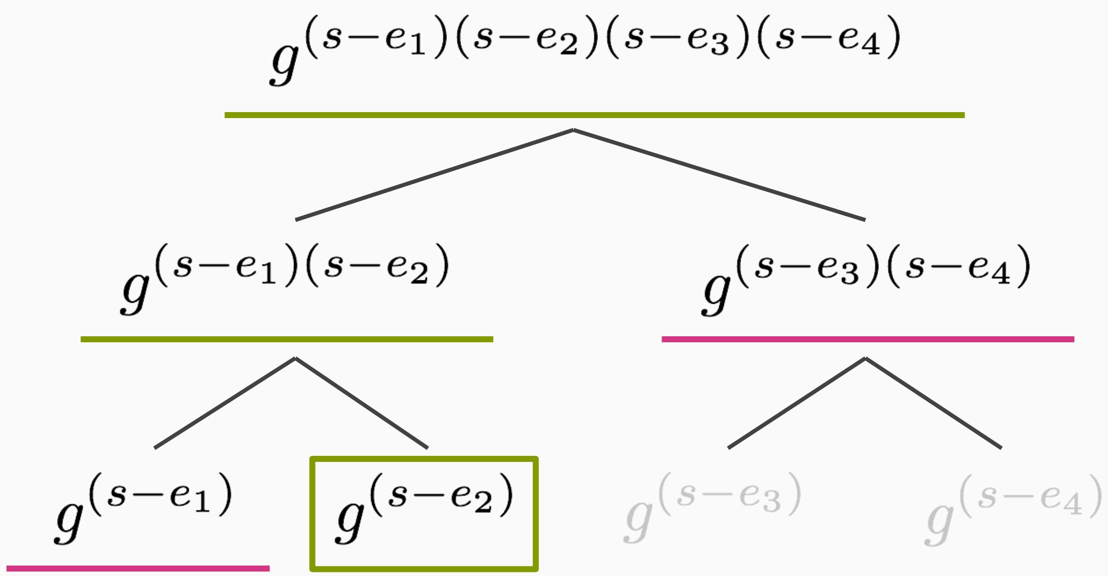
Bilinear Trees (BTs):
Precomputed membership proofs

$$g^{(s-e_1)(s-e_2)(s-e_3)(s-e_4)}$$

$$g^{(s-e_1)(s-e_2)} \qquad g^{(s-e_3)(s-e_4)}$$

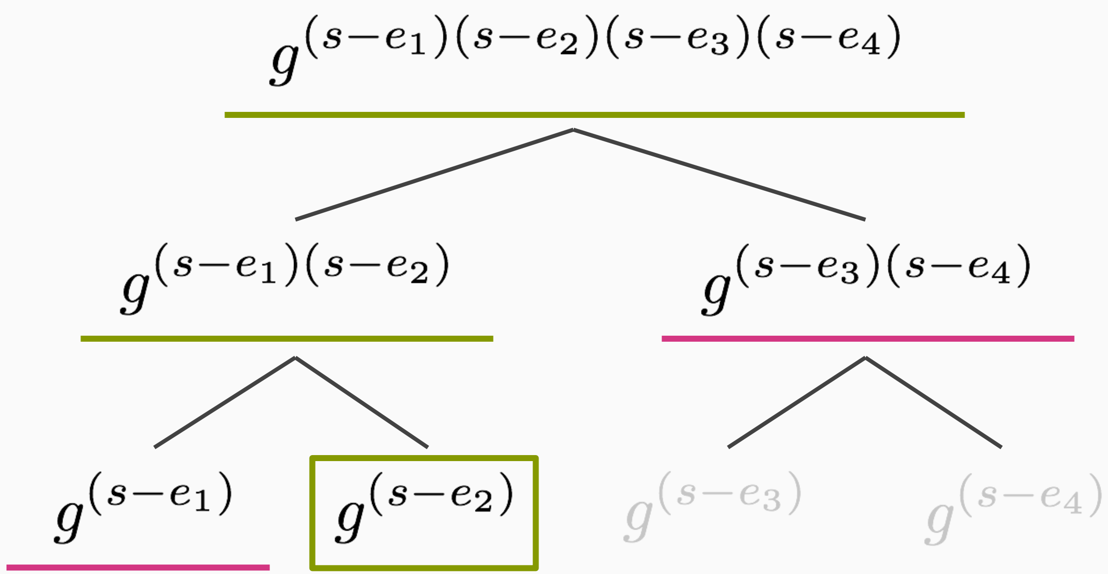
$$g^{(s-e_1)} \qquad g^{(s-e_2)} \qquad g^{(s-e_3)} \qquad g^{(s-e_4)}$$

Bilinear Trees (BTs):
Precomputed membership proofs



O(**n** log^2 **n**) time to precompute **all** membership proofs

Bilinear Trees (BTs):
Precomputed membership proofs



O(**n** log^2 **n**) time to precompute **all** membership proofs
...but what about precomputing non-membership?

The road so far...

1. Bilinear accumulators
2. Bilinear Trees (BTs)
3. **Bilinear Prefix Trees (BPTs)**
4. Bilinear Frontier Trees (BFTs)
5. Amortization
6. *From AAS to AAD (not in this talk)*

Bilinear Prefix Trees (BTs):
Towards precomputed **non-**membership proofs

Bilinear Prefix Trees (BTs):

Towards precomputed **non-**membership proofs

$$\text{e.g., } e_1 = 011 \Rightarrow \mathsf{pfx}(e_1) = \{\varepsilon, 0, 01, 011\}$$

Bilinear Prefix Trees (BTs):
Towards precomputed **non**-membership proofs

$\mathsf{pfx}(e_1)$ $\mathsf{pfx}(e_2)$ $\mathsf{pfx}(e_3)$ $\mathsf{pfx}(e_4)$

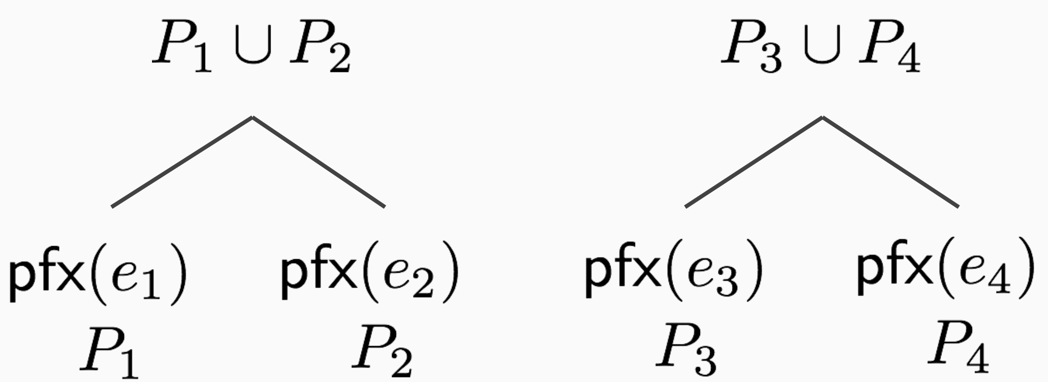
$$\text{e.g., } e_1 = 011 \Rightarrow \mathsf{pfx}(e_1) = \{\varepsilon, 0, 01, 011\}$$

Bilinear Prefix Trees (BTs):
Towards precomputed **non**-membership proofs

$\mathsf{pfx}(e_1)$	$\mathsf{pfx}(e_2)$	$\mathsf{pfx}(e_3)$	$\mathsf{pfx}(e_4)$
P_1	P_2	P_3	P_4

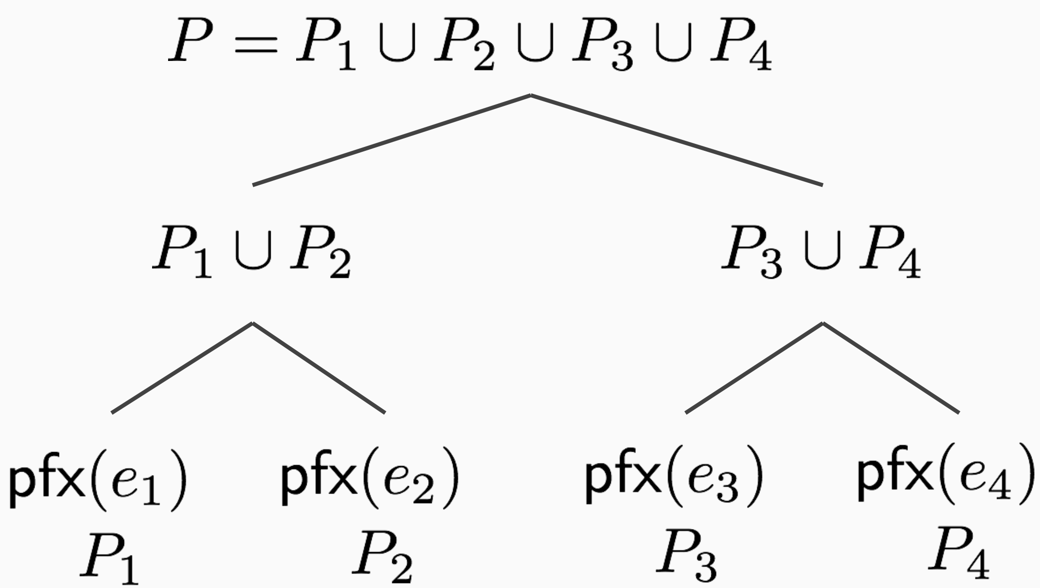
$$\text{e.g., } e_1 = 011 \Rightarrow \mathsf{pfx}(e_1) = \{\varepsilon, 0, 01, 011\}$$

Bilinear Prefix Trees (BTs):
Towards precomputed **non**-membership proofs



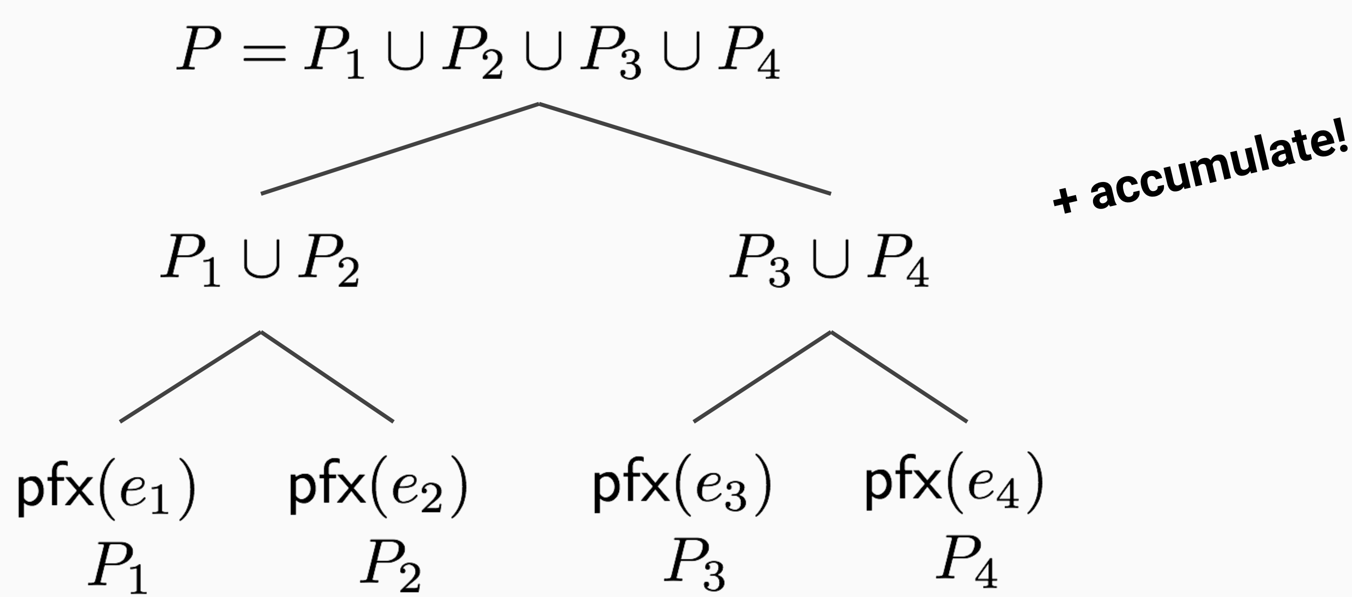
$$\text{e.g., } e_1 = 011 \Rightarrow \mathsf{pfx}(e_1) = \{\varepsilon, 0, 01, 011\}$$

Bilinear Prefix Trees (BTs):
Towards precomputed **non**-membership proofs



$$\text{e.g., } e_1 = 011 \Rightarrow \mathsf{pfx}(e_1) = \{\varepsilon, 0, 01, 011\}$$

Bilinear Prefix Trees (BTs):
Towards precomputed **non**-membership proofs



$$\text{e.g., } e_1 = 011 \Rightarrow \mathsf{pfx}(e_1) = \{\varepsilon, 0, 01, 011\}$$

Bilinear Prefix Trees (BTs):
Towards precomputed **non**-membership proofs

set of all prefixes!
i.e., a prefix tree

$$P = P_1 \cup P_2 \cup P_3 \cup P_4$$

$$P_1 \cup P_2 \qquad P_3 \cup P_4$$

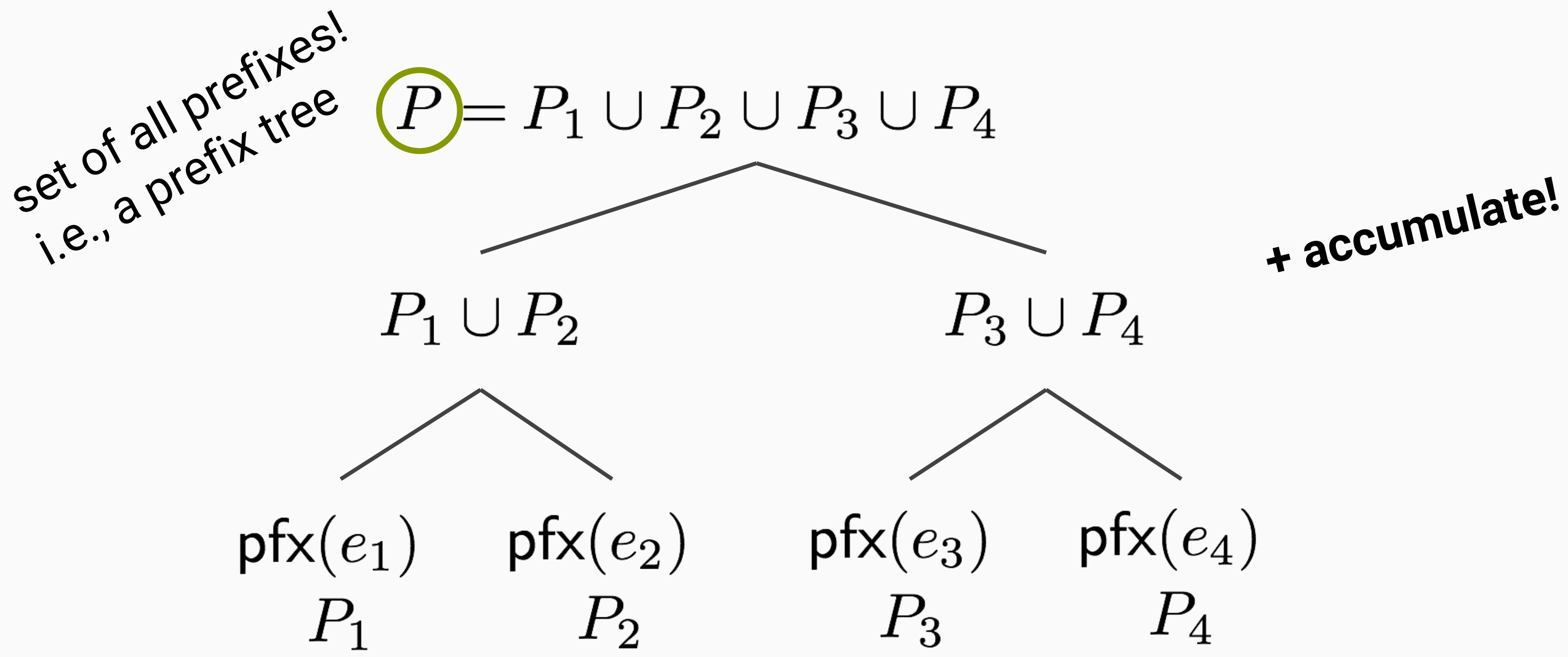
$$\mathsf{pfx}(e_1) \qquad \mathsf{pfx}(e_2) \qquad \mathsf{pfx}(e_3) \qquad \mathsf{pfx}(e_4)$$

$$P_1 \qquad P_2 \qquad P_3 \qquad P_4$$

+ accumulate!

$$\text{e.g., } e_1 = 011 \Rightarrow \mathsf{pfx}(e_1) = \{\varepsilon, 0, 01, 011\}$$

Bilinear Prefix Trees (BTs):
Towards precomputed **non**-membership proofs



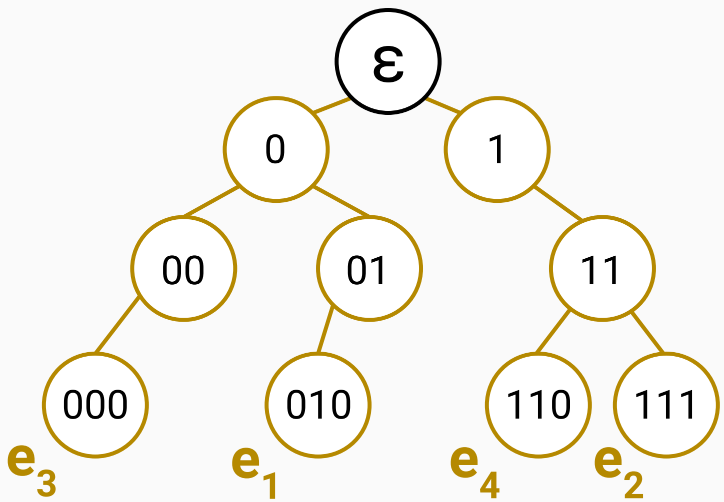
O(**λn** log^2 **n**) time to precompute **all** membership proofs

No seriously, how do we precompute non-membership?

The road so far...

1. Bilinear accumulators
2. Bilinear Trees (BTs)
3. Bilinear Prefix Trees (BPTs)
4. **Bilinear Frontier Trees (BFTs)**
5. Amortization
6. *From AAS to AAD (not in this talk)*

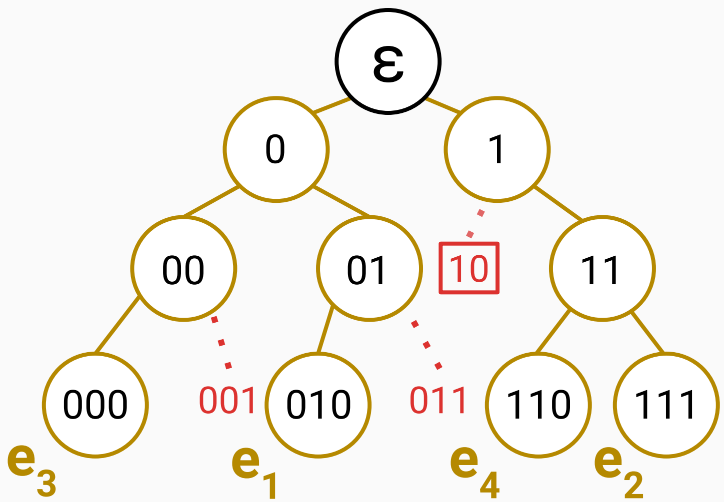
Bilinear Frontier Trees (BFTs)
Precompute non-membership proofs



P = prefixes of {e_1, e_2, e_3, e_4}

= set in root of BPT

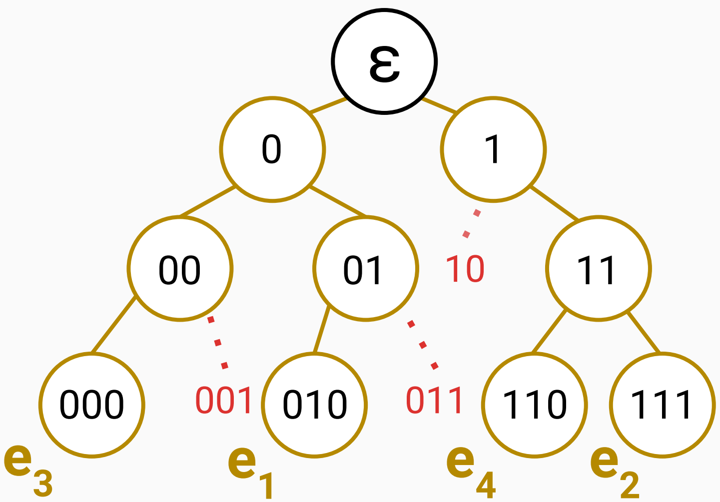
Bilinear Frontier Trees (BFTs)
Precompute non-membership proofs



P = prefixes of {e_1, e_2, e_3, e_4}

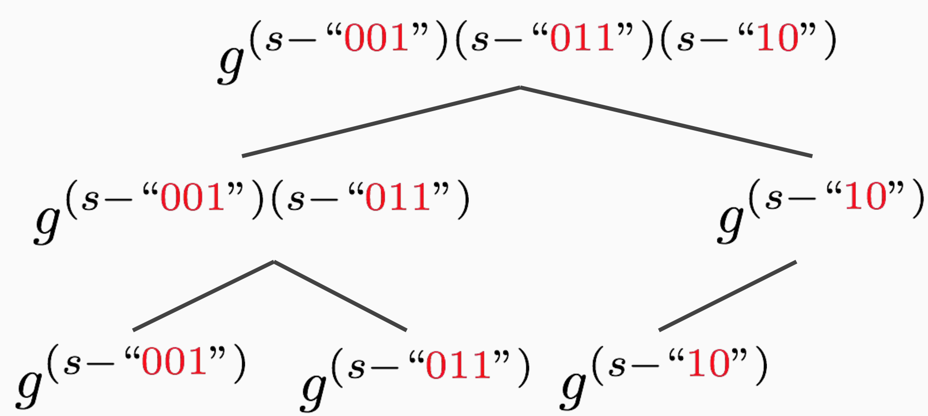
= set in root of BPT

Bilinear Frontier Trees (BFTs)
Precompute non-membership proofs



P = prefixes of {e_1, e_2, e_3, e_4}

= set in root of BPT



F = frontier(**P**)

F ∩ **P** = **∅**

disjointness proof!

O(**λn** log^2 **n**) time to precompute **all non-**membership proofs

The road so far...

1. Bilinear accumulators
2. Bilinear Trees (BTs)
3. Bilinear Prefix Trees (BPTs)
4. Bilinear Frontier Trees (BFTs)
5. **Amortization**
6. *From AAS to AAD (not in this talk)*

Dynamic AAS via amortization

Static AAS data structure so far. How can we append efficiently? And what about *append-only proofs*?

Dynamic AAS via amortization

Static AAS data structure so far. How can we append efficiently? And what about *append-only proofs*?

acc($\mathbf{E_1}$)

$\mathbf{E_i}$ = pfx(e_i)

Dynamic AAS via amortization

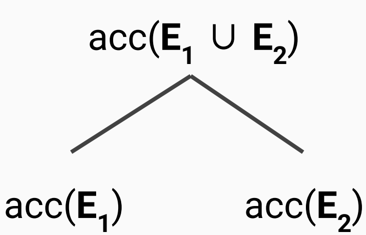
Static AAS data structure so far. How can we append efficiently? And what about *append-only proofs*?

acc($\mathbf{E}_1$) acc($\mathbf{E}_2$)

$\mathbf{E}_i$ = pfx(e_i)

Dynamic AAS via amortization

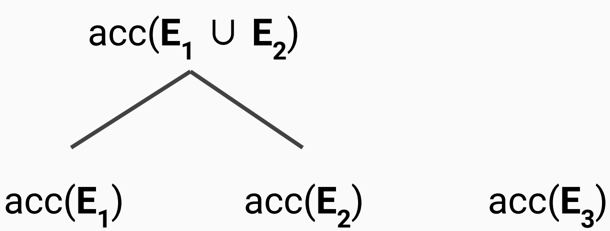
Static AAS data structure so far. How can we append efficiently? And what about *append-only proofs*?



$\mathbf{E}_i$ = pfx(e_i)

Dynamic AAS via amortization

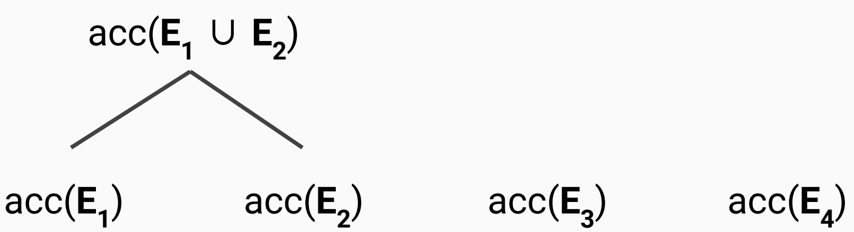
Static AAS data structure so far. How can we append <u>efficiently</u>? And what about *append-only proofs*?



$\mathbf{E_i} = pfx(e_i)$

Dynamic AAS via amortization

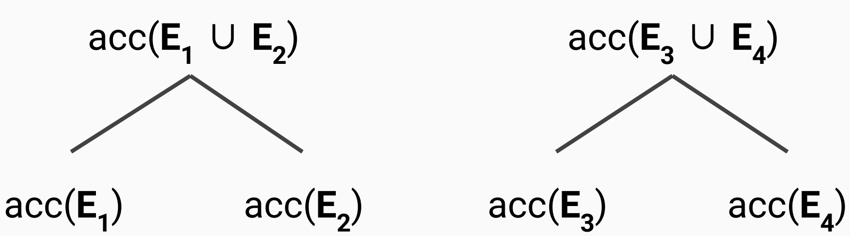
Static AAS data structure so far. How can we append efficiently? And what about *append-only proofs*?



$\mathbf{E_i}$ = pfx(e_i)

Dynamic AAS via amortization

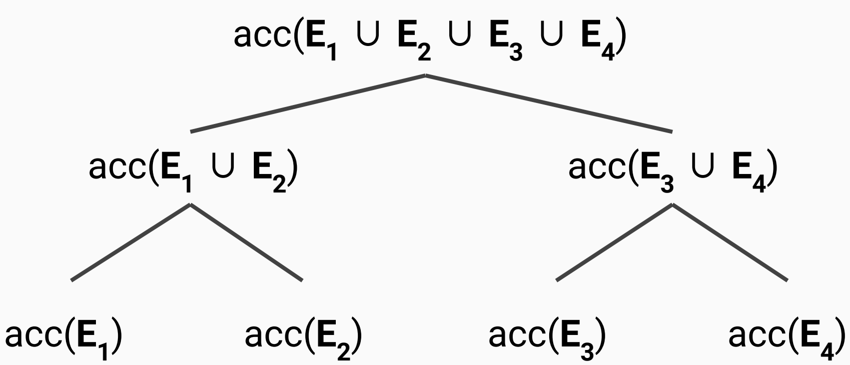
Static AAS data structure so far. How can we append <u>efficiently</u>? And what about *append-only proofs*?



$\mathbf{E_i}$ = pfx(e_i)

Dynamic AAS via amortization

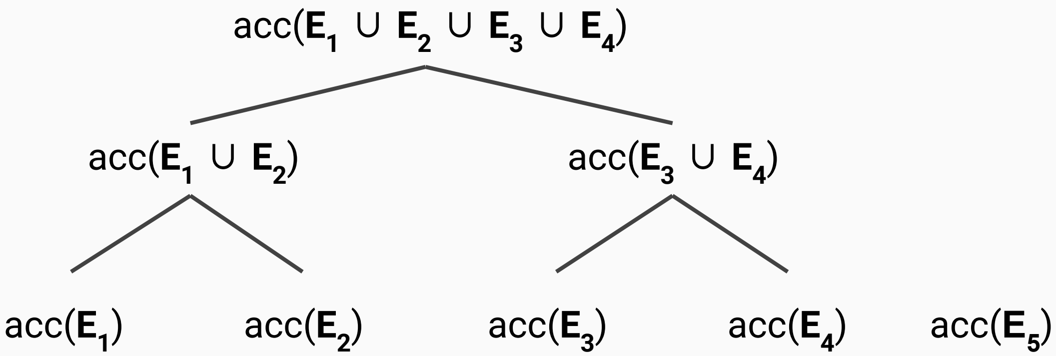
Static AAS data structure so far. How can we append efficiently? And what about *append-only proofs*?



$\mathbf{E}_i = \mathrm{pfx}(e_i)$

Dynamic AAS via amortization

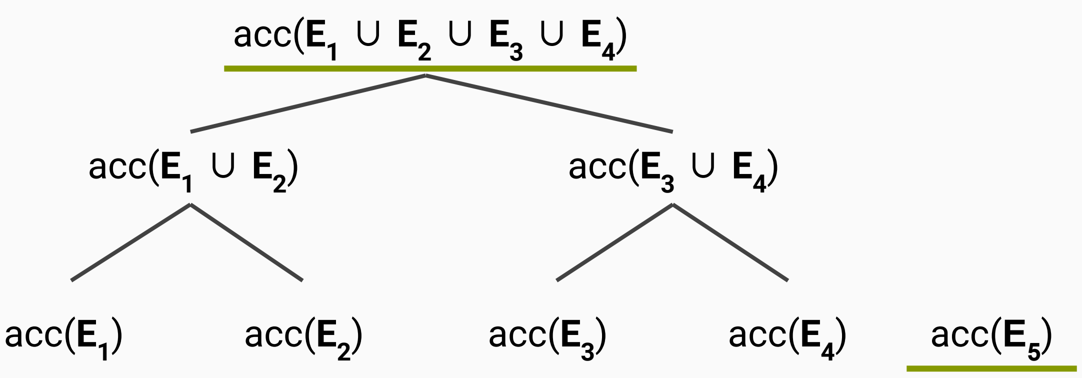
Static AAS data structure so far. How can we append efficiently? And what about *append-only proofs*?



$\mathbf{E_i}$ = pfx(e_i)

Dynamic AAS via amortization

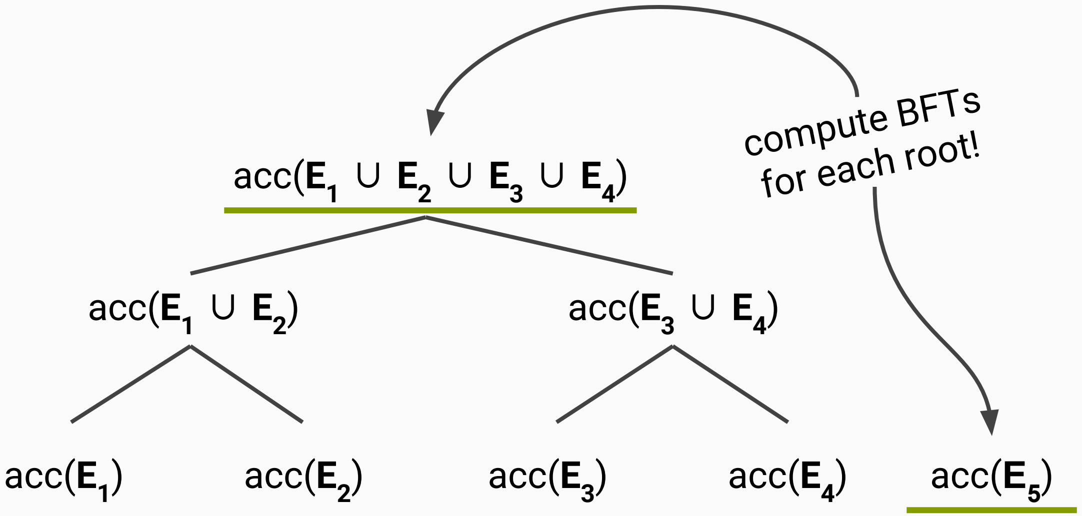
Static AAS data structure so far. How can we append <u>efficiently</u>? And what about *append-only proofs*?



$\mathbf{E_i}$ = pfx(e_i)

Dynamic AAS via amortization

Static AAS data structure so far. How can we append efficiently? And what about *append-only proofs*?

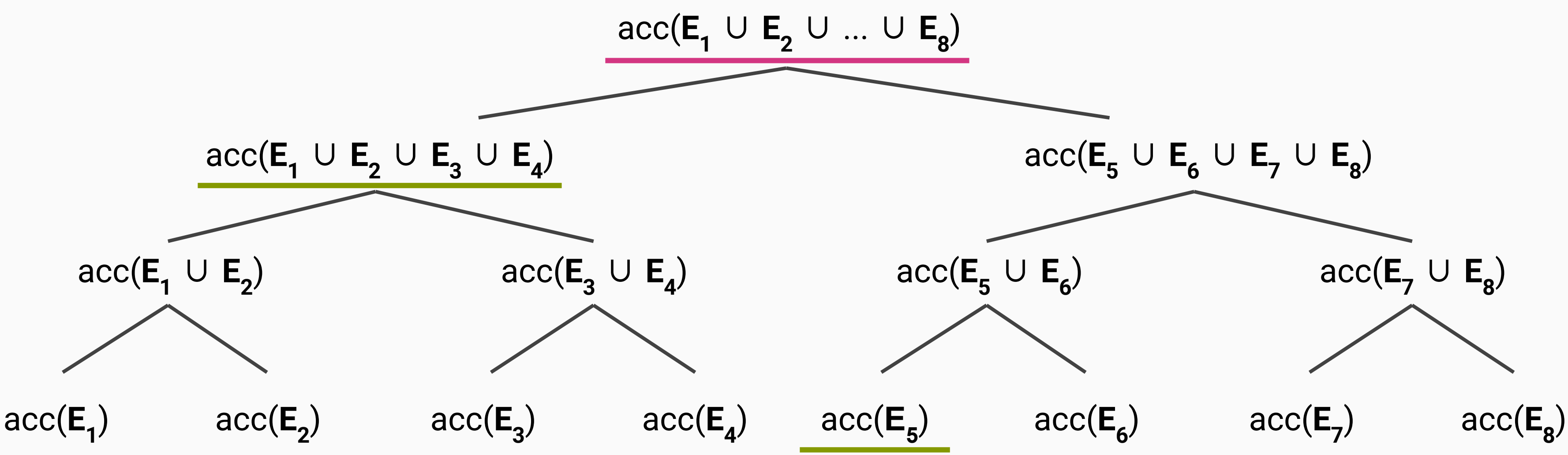


$\mathbf{E_i}$ = pfx(e_i)

Dynamic AAS via amortization

$T(\boldsymbol{\lambda}, \mathbf{n}) = 2T(\boldsymbol{\lambda}, \mathbf{n}/2) + O(\boldsymbol{\lambda}\mathbf{n} \log^2 \mathbf{n}) = O(\boldsymbol{\lambda}\mathbf{n} \log^3 \mathbf{n})$

$\Rightarrow O(\lambda \log^3 \mathbf{n})$ *amortized* append time



The road so far...

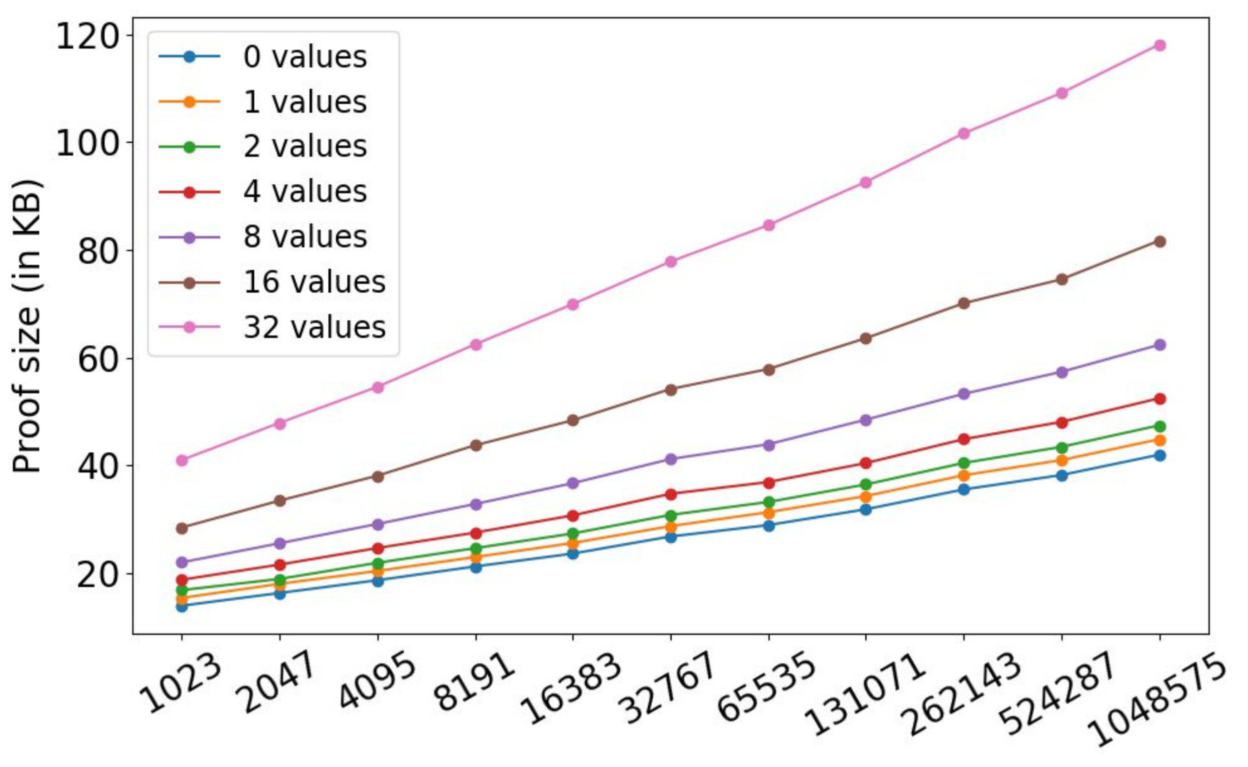
1. Bilinear accumulators
2. Bilinear Trees (BTs)
3. Bilinear Prefix Trees (BPTs)
4. Bilinear Frontier Trees (BFTs)
5. Amortization
6. *From AAS to AAD (not in this talk)*

AAD from AAS

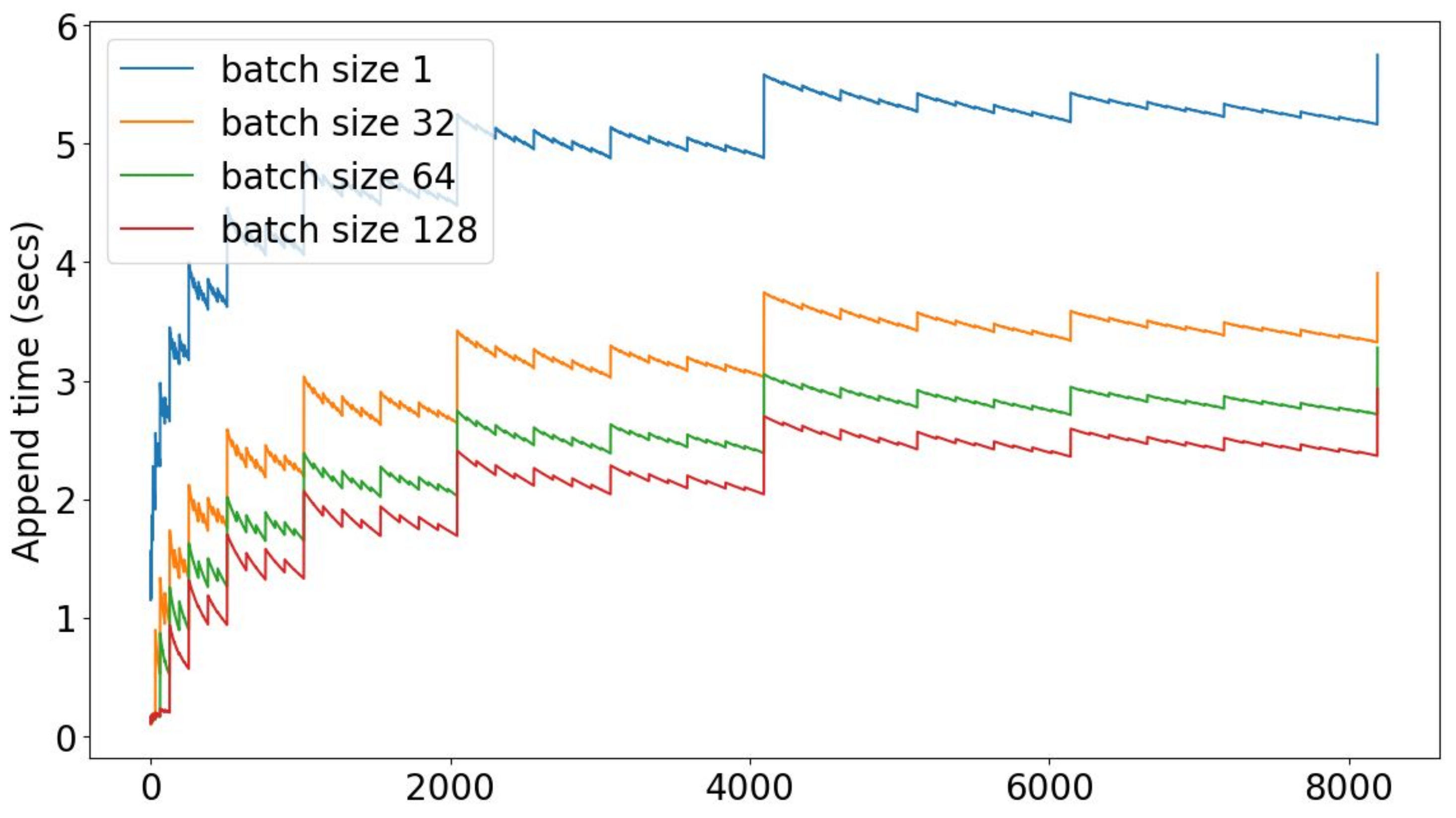
Quick idea: Build AAS over H(**k**) | H(**v**).

Plus, leverage frontier nodes for lookup proofs.

Experiments: Lookup proof size



Experiments: Append time



Conclusion

- HTTPs is vulnerable to CA compromises
- Certificate Transparency (CT) helps *detect* CA compromises
  - ...but CT logs are inefficient to audit
- We introduced **Append-only Authenticated Dictionaries (AADs)**
  - Foundation for building efficient-to-audit transparency logs
  - 200x bandwidth savings
  - Further secure HTTPs and messaging apps (e.g., WhatsApp)
- Future work
  - Faster appends (de-amortization?)
  - Smaller lookups (SNARKs?)
  - Simpler assumptions?

Appendix

Abstracting Certificate Transparency (CT)

An **authenticated dictionary**. Maps key to list of values. (i.e., a *domain name* to its *history of certificates*). Once value is added to key, it cannot be removed.

Abstracting Certificate Transparency (CT)

An **authenticated dictionary**. Maps key to list of values. (i.e., a *domain name* to its *history of certificates*). Once value is added to key, it cannot be removed.

Server API (i.e., log server)**:**

Abstracting Certificate Transparency (CT)

An **authenticated dictionary**. Maps key to list of values. (i.e., a *domain name* to its *history of certificates*). Once value is added to key, it cannot be removed.

Server API (i.e., log server):

$$\text{Append}(\mathcal{D}_i, k, v) \rightarrow d_{i+1}, \mathcal{D}_{i+1} = \mathcal{D}_i \cup \{(k, v)\}$$

Abstracting Certificate Transparency (CT)

An **authenticated dictionary**. Maps key to list of values. (i.e., a *domain name* to its *history of certificates*). Once value is added to key, it cannot be removed.

Server API (i.e., log server):

$$\mathsf{Append}(\mathcal{D}_i, k, v) \rightarrow d_{i+1}, \mathcal{D}_{i+1} = \mathcal{D}_i \cup \{(k, v)\}$$

$$\mathsf{ProveLookup}(\mathcal{D}_i, k) \rightarrow \pi_V, V = \{v_1, v_2, \ldots, v_m\}$$

Abstracting Certificate Transparency (CT)

An **authenticated dictionary**. Maps key to list of values. (i.e., a *domain name* to its *history of certificates*). Once value is added to key, it cannot be removed.

Server API (i.e., log server):

$$\text{Append}(\mathcal{D}_i, k, v) \rightarrow d_{i+1}, \mathcal{D}_{i+1} = \mathcal{D}_i \cup \{(k, v)\}$$

$$\text{ProveLookup}(\mathcal{D}_i, k) \rightarrow \pi_V, V = \{v_1, v_2, \ldots, v_m\}$$

$$\text{ProveAppendOnly}(\mathcal{D}_i, \mathcal{D}_j) \rightarrow \pi_{i,j}$$

Abstracting Certificate Transparency (CT)

An **authenticated dictionary**. Maps key to list of values. (i.e., a *domain name* to its *history of certificates*). Once value is added to key, it cannot be removed.

Server API (i.e., log server):

$$\mathsf{Append}(\mathcal{D}_i, k, v) \rightarrow d_{i+1}, \mathcal{D}_{i+1} = \mathcal{D}_i \cup \{(k, v)\}$$

$$\mathsf{ProveLookup}(\mathcal{D}_i, k) \rightarrow \pi_V, V = \{v_1, v_2, \ldots, v_m\}$$

$$\mathsf{ProveAppendOnly}(\mathcal{D}_i, \mathcal{D}_j) \rightarrow \pi_{i,j}$$

Client API (e.g., VISA, Alice):

Abstracting Certificate Transparency (CT)

An **authenticated dictionary**. Maps key to list of values. (i.e., a *domain name* to its *history of certificates*). Once value is added to key, it cannot be removed.

Server API (i.e., log server):

$$\mathsf{Append}(\mathcal{D}_i, k, v) \rightarrow d_{i+1}, \mathcal{D}_{i+1} = \mathcal{D}_i \cup \{(k, v)\}$$

$$\mathsf{ProveLookup}(\mathcal{D}_i, k) \rightarrow \pi_V, V = \{v_1, v_2, \ldots, v_m\}$$

$$\mathsf{ProveAppendOnly}(\mathcal{D}_i, \mathcal{D}_j) \rightarrow \pi_{i,j}$$

Client API (e.g., VISA, Alice):

$$\mathsf{VerLookup}(d_i, k, V, \pi_V) \rightarrow 0/1$$

Abstracting Certificate Transparency (CT)

An **authenticated dictionary**. Maps key to list of values. (i.e., a *domain name* to its *history of certificates*). Once value is added to key, it cannot be removed.

Server API (i.e., log server):

$$\mathsf{Append}(\mathcal{D}_i, k, v) \rightarrow d_{i+1}, \mathcal{D}_{i+1} = \mathcal{D}_i \cup \{(k, v)\}$$

$$\mathsf{ProveLookup}(\mathcal{D}_i, k) \rightarrow \pi_V, V = \{v_1, v_2, \ldots, v_m\}$$

$$\mathsf{ProveAppendOnly}(\mathcal{D}_i, \mathcal{D}_j) \rightarrow \pi_{i,j}$$

Client API (e.g., VISA, Alice):

$$\mathsf{VerLookup}(d_i, k, V, \pi_V) \rightarrow 0/1$$

Completeness guarantees!

Abstracting Certificate Transparency (CT)

An **authenticated dictionary**. Maps key to list of values. (i.e., a *domain name* to its *history of certificates*). Once value is added to key, it cannot be removed.

Server API (i.e., log server):

$$\mathsf{Append}(\mathcal{D}_i, k, v) \rightarrow d_{i+1}, \mathcal{D}_{i+1} = \mathcal{D}_i \cup \{(k, v)\}$$

$$\mathsf{ProveLookup}(\mathcal{D}_i, k) \rightarrow \pi_V, V = \{v_1, v_2, \ldots, v_m\}$$

$$\mathsf{ProveAppendOnly}(\mathcal{D}_i, \mathcal{D}_j) \rightarrow \pi_{i,j}$$

Client API (e.g., VISA, Alice):

$$\mathsf{VerLookup}(d_i, k, V, \pi_V) \rightarrow 0/1$$

$$\mathsf{VerAppendOnly}(d_i, d_j, \pi_{i,j}) \rightarrow 0/1$$

Completeness guarantees!

Abstracting Certificate Transparency (CT)

An **authenticated dictionary**. Maps key to list of values. (i.e., a *domain name* to its *history of certificates*). Once value is added to key, it cannot be removed.

Server API (i.e., log server):

$$\mathsf{Append}(\mathcal{D}_i, k, v) \rightarrow d_{i+1}, \mathcal{D}_{i+1} = \mathcal{D}_i \cup \{(k, v)\}$$

$$\mathsf{ProveLookup}(\mathcal{D}_i, k) \rightarrow \pi_V, V = \{v_1, v_2, \ldots, v_m\}$$

$$\mathsf{ProveAppendOnly}(\mathcal{D}_i, \mathcal{D}_j) \rightarrow \pi_{i,j}$$

Client API (e.g., VISA, Alice):

$$\mathsf{VerLookup}(d_i, k, V, \pi_V) \rightarrow 0/1$$

$$\mathsf{VerAppendOnly}(d_i, d_j, \pi_{i,j}) \rightarrow 0/1$$

Completeness guarantees!

Gap between **i** and **j** is typically large.