Trust Anchor Negotiation

Bob Beck
David Benjamin
Devon O'Brien

Trust Anchor Negotiation

Goals and motivation

- https://github.com/davidben/tls-trust-expressions/blob/main/explainer.md
- https://github.com/davidben/tls-trust-expressions/blob/main/pki-transition-strategies.md

Alternate approach: TLS Trust Anchor Identifiers

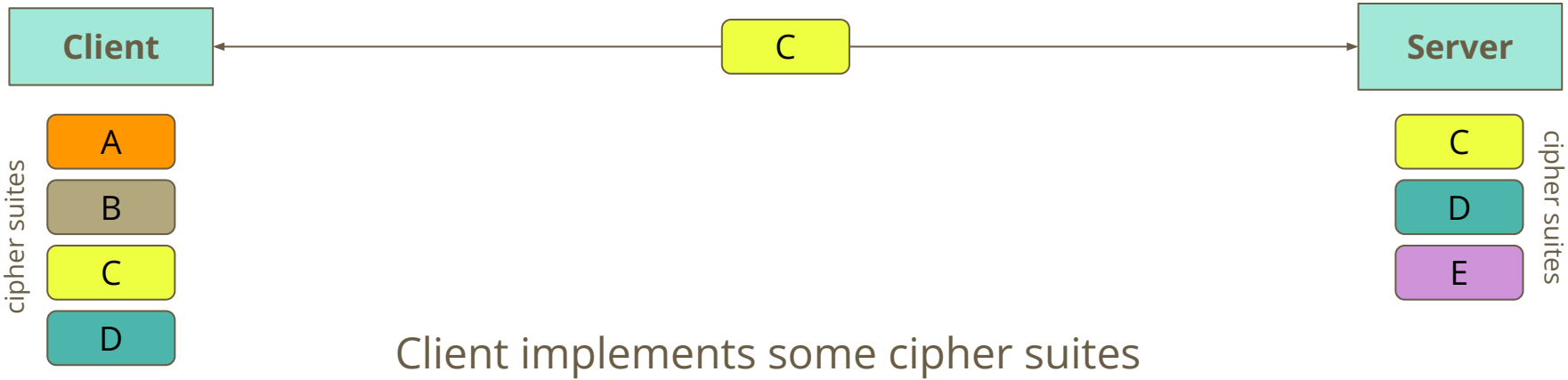
- draft-beck-tls-trust-anchor-ids

Security considerations

- https://davidben.github.io/tls-trust-expressions/draft-beck-tls-trust-anchor-ids.html#name-security-considerations
- https://davidben.github.io/tls-trust-expressions/draft-davidben-tls-trust-expr.html#name-security-considerations

Next steps

Recap: TLS Parameter Negotiation

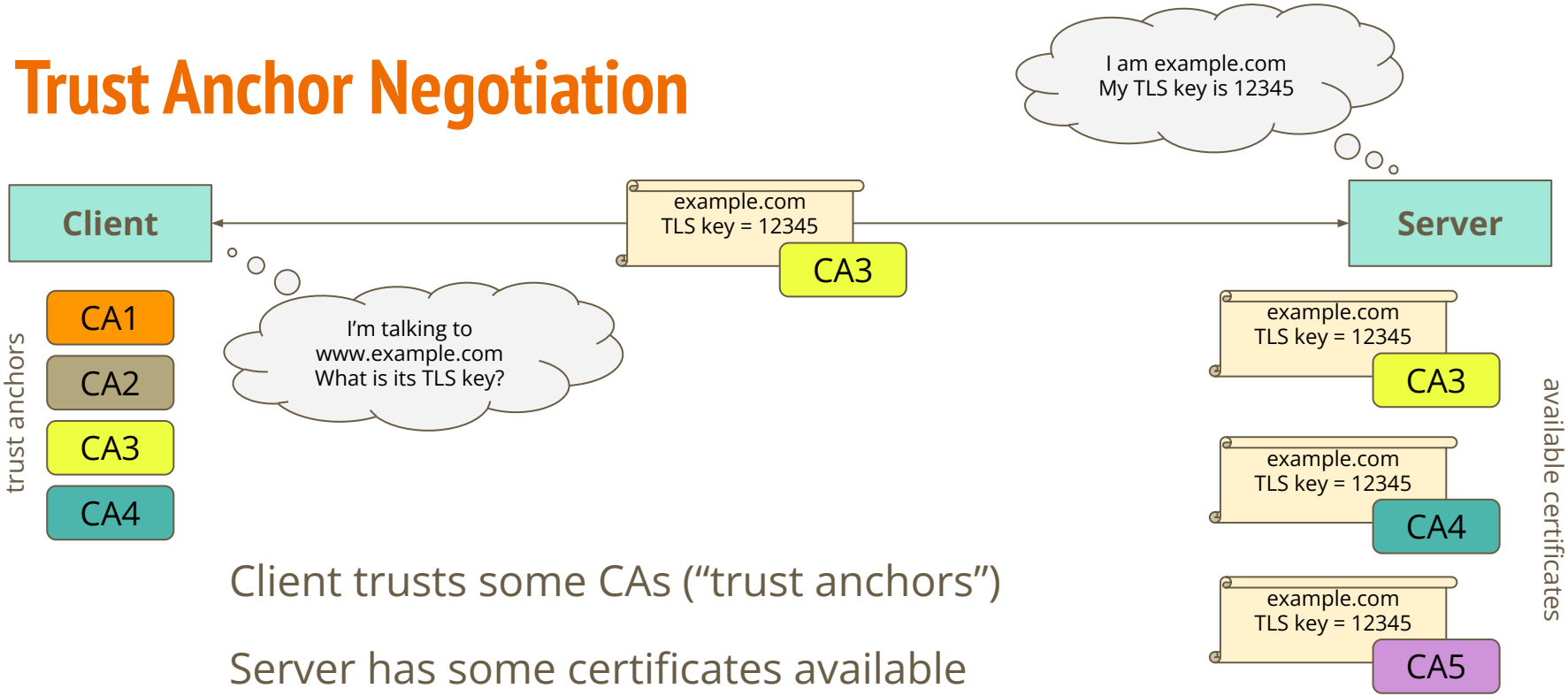


Client implements some cipher suites

Server implements some cipher suites

***Goal**: Select best cipher suite in common*

Trust Anchor Negotiation



Client trusts some CAs ("trust anchors")

Server has some certificates available

***Goal**: Select certificate based on what client trusts*

Client Diversity

TLS client ecosystems are already diverse

Root programs in the “same” PKI make independent decisions

Older and newer clients diverge over time as PKIs evolve

Constrained devices (e.g. IoT) often cannot update at all

Some clients (e.g. mobile apps) pin to particular CAs or keys

Servers must somehow navigate this

PKI Agility

TLS client diversity can be useful

Old and new clients diverge as PKIs change, but PKI changes are necessary for user security

- Rotating root keys
- Removing CAs
- Adding CAs

Performance — avoid lowest common denominator

- Up-to-date clients can trust short-lived “intermediate” directly
- Avoid sending unnecessary cross-signs when CA is already trusted
- Parallel issuance when cross-signs are expensive or not viable
- Even more tailored designs (e.g. Merkle Tree Certificates)

https://github.com/davidben/tls-trust-expressions/blob/main/pki-transition-strategies.md

Security vs Availability

If servers cannot meet diverse client needs, **security and availability conflict:**

Security — PKI changes are sometimes necessary for user security

Availability — server must satisfy all supported clients

Example: Client removes CA1, supports CA2. Server *also* needs to support other clients which *only* support CA1.

When availability and security conflict, availability usually wins

~~*Security wins* — servers drop support for some clients~~

Availability wins — PKI changes needed for user security do not happen or are delayed

User security is the casualty of this conflict

Negotiation gives another option: servers use CA1 *and* CA2, depending on client

Trust Anchor Negotiation

Enable servers to handle client diversity…

…so server availability does not conflict with PKI security and performance

PKI transitions do not impede overall server availability

Root programs can make timely security decisions on behalf of users

Minimize operational burden for server operators

- More server operators than CAs and root programs
- ACME extensions for CAs to send multiple certificates
- Prefer fewer mechanisms that cover the whole problem

Minimize bytes on the wire, especially as we transition to PQC

https://github.com/davidben/tls-trust-expressions/blob/main/explainer.md

Two Approaches

Problem: `certificate_authorities` (RFC 8446) is too large

- X.509 names are large (average 100 bytes per name)
- Some PKIs have many CAs (hundreds)

Two approaches

- Trust Expressions (draft-davidben-tls-trust-expr)
- Trust Anchor IDs (draft-beck-tls-trust-anchor-ids)

Recap: Trust Expressions

Clients express CA lists relative to named “trust stores”

- Trust stores are maintained by root programs
- Client can express deltas, but not as flexible for fully arbitrary CA lists

Prioritizes minimizing server operator burden

- Static server configuration, ideally from CA

Cost: Continuous coordination between root programs and CAs

- Most of the complexity is here

draft-davidben-tls-trust-expr

Trust Anchor IDs: Short CA Names

Problem: X.509 names are large (around 100 bytes)

Solution: Make shorter names!

1. Allocate a *trust anchor ID* for each CA
    - We used relative OIDs under Private Enterprise Numbers (RFC 9371)
    - About 5 bytes per CA
2. Configure in server with CertificatePropertyList from CA
3. Configure in client with other trust anchor information from root program
4. Define new `trust_anchors` TLS extension, like `certificate_authorities` but carries trust anchor IDs instead

Trust Anchor IDs: Server Offer, Client Select

Problem: Still cannot send full trust anchor list—client bandwidth and privacy constraints

Solution: Use an ECH-style retry after server offers its list of trust anchors —available server certificates are fewer and less privacy sensitive

1. Client MAY send empty trust anchor list, or some subset of its full list.
2. Server sends trust anchors of available certificates in EncryptedExtensions
3. On mismatch or error, client selects trust anchor it would accept from EncryptedExtensions
4. Client retries with new connection, sending selection in trust anchor list

Alternate: Design a retry flow in the same handshake

Trust Anchor IDs: Push to DNS

Problem: Retry flow adds latency

Solution: List available server certificates in DNS

1. Define `tls-trust-anchors` SvcParam for HTTPS/SVCB records
2. Client uses SvcParam for initial prediction in `trust_anchors` extension
3. Repair mispredictions (e.g. stale DNS) with retry flow

Server operator challenge: keep DNS and TLS configuration in sync

Similar challenge as ECH and key share prediction

Use draft-ietf-tls-wkech

Trust Expressions	Trust Anchor IDs
Applicability	
Only expresses CAs lists relative to root program “trust stores”	Supports arbitrary CA lists
Applies analogously to client certificates	Short CA names apply to client and server certificates Retry is only possible with server certificates.
Deployment	
Servers use static configuration from CAs	Servers need to synchronize TLS and DNS for best performance
CAs and root programs need continuous coordination via “trust store manifests”	CAs and root programs use static configuration
Clients use static configuration from root programs	Clients need logic to fetch information from DNS and implement retry
Privacy	
Client CA lists are passively observable	Client CA lists require active probing to enumerate
Service’s available certificates require active probing to enumerate	Trust anchors of service’s available certificates are passively observable

Security Considerations

We have enumerated the various scenarios discussed on-list

- Technical considerations have been added to the drafts[1][2]
- Policy consideration/speculation have been added to a supporting document[3]
- We have performed an initial analysis of discussion points

If there are missing scenarios, please file an issue so we can add them

If there is additional analysis needed, please discuss on-list or file an issue

[1] https://github.com/davidben/tls-trust-expressions/blob/main/draft-beck-tls-trust-anchor-ids.md#security-considerations

[2] https://github.com/davidben/tls-trust-expressions/blob/main/draft-davidben-tls-trust-expr.md#security-considerations

[3] https://github.com/davidben/tls-trust-expressions/blob/main/surveillance-and-trust-anchor-negotiation.md

TLS Trust Anchor Negotiation

Two proposed approaches to address PKI transitions in TLS

- draft-davidben-tls-trust-expr, draft-beck-tls-trust-anchor-ids
- If you have read either draft, **thank you!**

We'd like feedback from WG on preferred approach

Other feedback also welcome on list or GitHub: https://github.com/davidben/tls-trust-expressions/issues

Next steps

- Discussion, hums for preferred approach, call for adoption