Designing and Verifying a Post-Quantum Protocol for Privacy Critical Applications using ABE and ProVerif

Robert-William Evans & Florian Kammueller

Middlesex University

DPM 2026

September 17, 2026

Motivation

- Quantum computers threaten digital security
- Harvest now, decrypt later attacks makes the quantum threat more pronounced
- Data with long-term value is at risk
- Enterprises need to share data selectively.

Research Question 1

Can attribute-based access control be integrated into a protocol through the straightforward combination of post-quantum cryptographic primitives, while remaining resistant to quantum attacks?

Research Question 2

Which security properties of such a protocol can be formally verified using ProVerif?

Post-Quantum Building Blocks

- ML-DSA: post-quantum digital signature scheme (NIST-standardised, 2024)
- ML-KEM: post-quantum key encapsulation (formerly Kyber), establishes a shared key without sending it over the network
- CoverCrypt: ETSI-standardised, post-quantum attribute-based encryption scheme, from the CP-ABE family

Attribute-Based Access Control

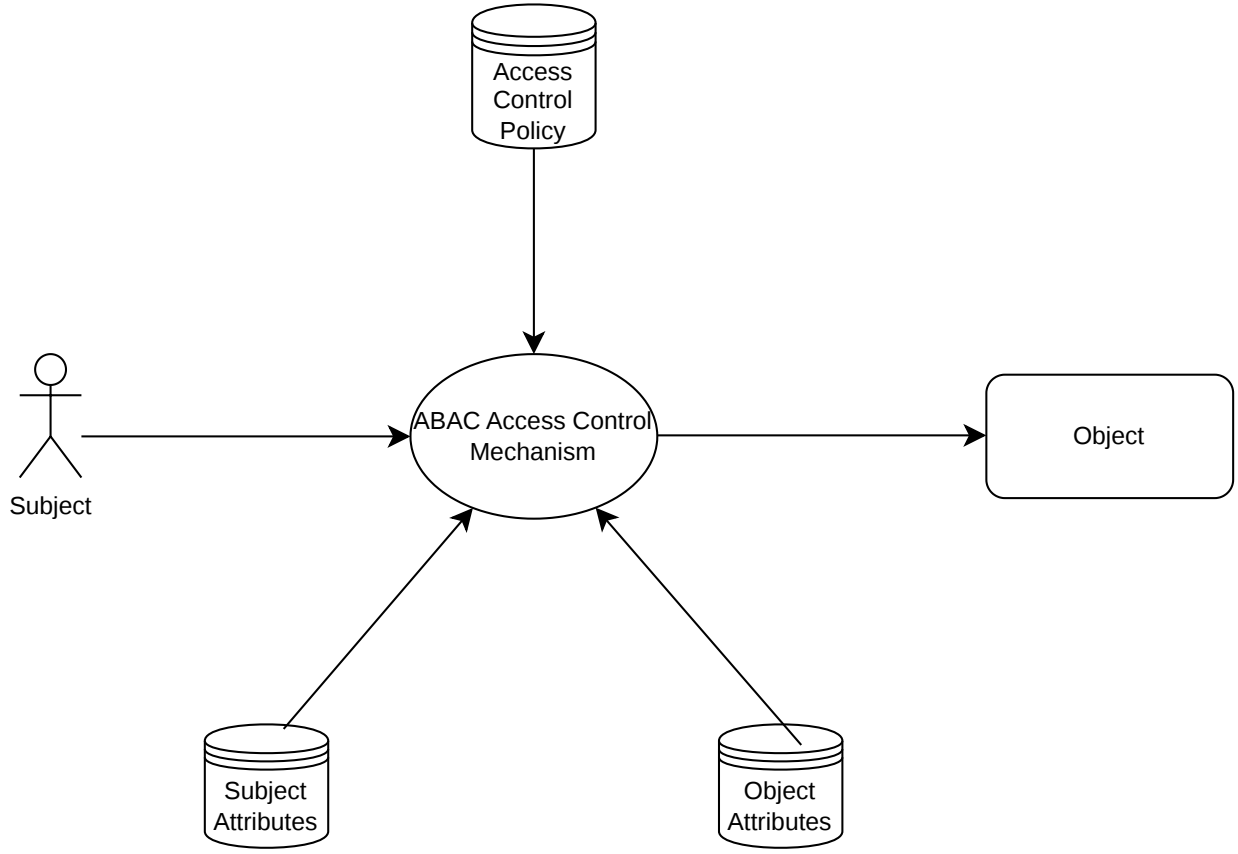

Figure 1: ABAC model

Attribute-Based Encryption

- ABE embeds the access control mechanism directly into the cryptography
- CoverCrypt (CP-ABE): the **encryption policy** is attached to the ciphertext; the **access policy**, derived from the user's attributes, is used to generate their decryption key
- Note: Cosmian's terminology differs from standard CP-ABE usage. What is usually called the "access policy" (ciphertext-side) is CoverCrypt's **encryption policy**
- Case study: an enterprise with two departments (AI, Robotics) and two clearance levels (High, Low)

```
(Department::Robotics || Department::AI) && Clearance
                                    Level::High
```

Proverif

- Proverif is a trusted protocol verifier
- It works on the Dolev-Yao formal model.
- Attacker controls the network
- Cryptography is assumed to be perfect.
- Events are annotations that mark important points during protocol execution
- Properties to be validated include mutual authentication, confidentiality and correctness of decryption

Upload Phase

$$
\begin{array}{lll}
1. & A \to S & : \quad (A,\ N_A) \\
2. & S \to A & : \quad \left(\{N_A, N_S\}_{K_S^{-1}},\ N_S\right) \\
3. & A \to S & : \quad \left(\{N_A, N_S\}_{K_A^{-1}},\ \{M, P_M\}_{mpk}\right)
\end{array}
$$

Figure 2: The protocol in Alice-Bob-notation

Events emitted during this stage include
`AliceAuthenticated(na,ns)` and
`ServerAuthenticated(na,ns)`

Download Phase

$$
\begin{array}{lllll}
4. & B \to S & : & (B,\ N_B) \\
5. & S \to B & : & \left(\{N_B, N'_S\}_{K_S^{-1}},\ N'_S\right) \\
6. & B \to S & : & \left(\{N_B, N'_S\}_{K_B^{-1}},\ C_{B,S},\ id_M\right) \\
7. & S \to B & : & \left(\{M\}_{K_{B,S}},\ \{\{\{M\}_{K_{B,S}}, N_B, N'_S\}_{K_S^{-1}}\}\right)
\end{array}
$$

Figure 3: The protocol in Alice-Bob-notation

Events emitted during this stage include
`DecryptionSucceeded(ap,ep)`

Verification Results for Confidentiality

A single secrecy query is enough: given everything observed between Alice, the Server, and Bob, the model checker cannot derive the secret document.

Example of a secrecy query

```
Query not attacker(secret_doc[]) is true.
```

Verification Results for Mutual Authentication

Injective correspondence queries confirm Alice<–> Server and Bob<–> Server mutual authentication, each tied to a unique prior event.

Examples of injective correspondence queries

```
Query inj-event(AliceAuthenticated(na,ns)) ==> inj-
    event(ServerAuthenticated(na,ns)) is true
```

```
Query inj-event(BobAuthenticated(nb,ns)) ==> inj-
    event(RServerAuthenticated(nb,ns)) is true
```

Verification Results for Correctness of decryption

Decryption must follow CoverCrypt's logic exactly: it succeeds only when a user's access policy satisfies the document's encryption policy.

Examples of reachability queries

```
Query not event(DecryptionSucceeded(and_policy(dept_ai,high),
                                    and_enc_policy(dept_ai,high)))
                                        is false.

Query not
    event(DecryptionSucceeded(and_policy(dept_robotics,low),
          and_enc_policy(dept_robotics,high))) is true.


Query not event(DecryptionSucceeded(and_policy(dept_ai,high),
                      compound_enc_policy(and_enc_policy(dept_ai,high),
                      and_enc_policy(dept_robotics,high)))) is
                          false.
```

Implementation

- Client built with React
- Backend built with Spring Boot (Java)
- Database: PostgreSQL, hosted on Render
- File storage: Amazon S3
- Initial version uses TLS as the protected channel; full post-quantum deployment would add IBM's ML-DSA and Open Quantum Safe's ML-KEM implementations

Conclusion

- Each post-quantum primitive plays a role in the protocol
- Confidentiality, correctness of decryption, and mutual authentication all formally verified in ProVerif
- Data is quantum-safe before it ever reaches storage
- For future works, resilience of the protocol could be investigated