Hackit & Run LLP Secure Contract Exchange Protocol

Assignment 1: Cryptography

Cybersecurity Coursework

Author: Arash

Date: 15 October 2025

Contents

Abstract

This report proposes and demonstrates a secure remote contract exchange protocol for Hackit & Run LLP (H&R), a UK-based firm of solicitors specialising in property transactions. The goal is to design a cryptographically sound, legally enforceable system for secure document exchange between three stakeholders: H&R, the seller's solicitor, and Mrs. Harvey (the buyer). The protocol ensures confidentiality, integrity, authentication, and non-repudiation using modern public key cryptography. Two operational cases are considered: (a) when H&R and the seller's solicitor have previously communicated securely, and (b) when they are interacting for the first time. The protocol integrates authenticated Elliptic Curve Diffie–Hellman (ECDH) for session establishment, Elliptic Curve Digital Signature Algorithm (ECDSA) for identity assurance, and AES-GCM for authenticated encryption. A full implementation in Python, using the `cryptography` library, validates the proposed design. The report concludes with a discussion on security guarantees, limitations, and future enhancements.

1. Introduction

Hackit & Run LLP (H&R) specialises in remote property transactions where parties may not meet physically to exchange or sign documents. With increased reliance on digital channels, ensuring the confidentiality and integrity of contracts is vital. Traditional email or PDF exchanges are vulnerable to interception, tampering, or repudiation.

To maintain trust, compliance with UK legal frameworks such as the Electronic Identification and Trust Services (eIDAS) regulation and the UK Electronic Communications Act 2000 is essential. The goal of this work is to design a cryptographic protocol that provides:

- Confidentiality of contracts in transit and storage
- Integrity protection against tampering
- Authentication of each party
- Non-repudiation of signatures
- Simplified re-authentication for previously known parties

This report details the system design, cryptographic selections, protocol flow, implementation, and a practical demonstration through executable Python code.

2. Threat Model and Assumptions

The following threat landscape is assumed:

- Attackers may intercept, modify, or replay messages in transit.
- An adversary could impersonate another party without robust authentication.
- Contract confidentiality must be preserved even if an intermediary channel is insecure.
- Denial-of-service attacks are outside this scope, as are insider threats.

Assumptions:

1. Each party holds a long-term ECDSA key pair stored securely by their respective solicitors.
2. Keys are managed under organisational PKI or trusted certification authority.
3. The initial exchange occurs over an authenticated medium or with public key verification.
4. Contracts are static digital files (e.g., PDFs or text) representing legally binding documents.

Two cases are modelled:

Case A: Repeated communication — an existing session key allows instant resumption without handshake.

Case B: First communication — a full authenticated key exchange (ECDH + signatures) is performed.

3. Algorithm Selection and Justification

3.1 Elliptic Curve Cryptography (ECC)

ECC provides equivalent security to RSA at smaller key sizes, yielding efficiency and faster computation. The NIST P-256 (SECP256R1) curve is used for both ECDSA and ECDH operations.

3.2 Authenticated ECDH (Ephemeral-Static)

For first-time communication, both parties create ephemeral key pairs and sign their public keys with their long-term private ECDSA keys. This prevents Man-in-the-Middle attacks and provides perfect forward secrecy.

3.3 AES-GCM for Confidentiality and Integrity

Advanced Encryption Standard in Galois/Counter Mode (AES-GCM) offers both encryption and message authentication in a single operation. A 128-bit key derived via HKDF from the ECDH shared secret ensures authenticity and confidentiality.

3.4 HKDF for Key Derivation

The shared ECDH secret is passed through the HMAC-based Key Derivation Function (HKDF) using SHA-256. This ensures uniform entropy distribution and prevents key reuse attacks.

3.5 ECDSA for Digital Signatures

ECDSA provides compact, verifiable signatures for contract approval. Mrs. Harvey uses her private key to sign the document's hash (SHA-256), which is then verified by H&R and the seller's solicitor.

4. Proposed Protocol Design

4.1 Entities

- **H&R (Law Firm):** Mediator between buyer and seller's solicitor.
- **Seller's Solicitor (SS):** Represents Mr. L.M. Facey.
- **Buyer (B):** Mrs. Harvey, who signs the contract.

4.2 Protocol Overview

1. SS sends contract to H&R (encrypted with session key).
2. H&R forwards decrypted contract to Buyer.
3. Buyer signs the document hash + timestamp using ECDSA.
4. H&R verifies signature, re-encrypts using session key, and forwards to SS.

4.3 First-time Communication (Case B)

1. Both parties generate ephemeral EC key pairs.
2. Each signs its ephemeral public key with its long-term ECDSA key.
3. They exchange {ephemeral_pub, signature, certificate}.
4. On verification, they compute a shared secret via ECDH.
5. HKDF derives a 128-bit AES session key.
6. The session key is stored securely for reuse.

4.4 Repeated Communication (Case A)

1. Session key from prior exchange is reused.
2. Encrypted communication proceeds without handshake.

3. Key rotation is enforced periodically for forward secrecy.

4.5 Sequence Diagram

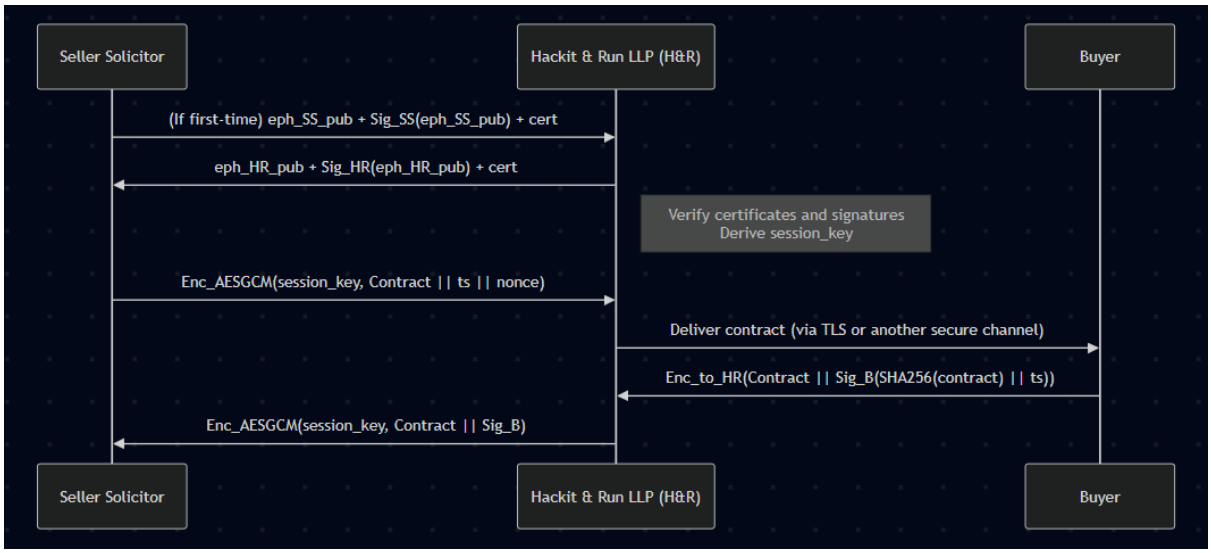

Figure 4.1: Sequence diagram of secure contract exchange protocol.

4.6 Pseudocode Summary

Listing 4.1: Simplified pseudocode of the protocol

```
if session_key_exists():
    reuse_session_key()
else:
    eph_hr, sig_hr = generate_signed_ephemeral(hr_priv)
    eph_ss, sig_ss = generate_signed_ephemeral(ss_priv)
    verify_signatures(eph_hr, eph_ss)
    shared_secret = ECDH(eph_hr_priv, eph_ss_pub)
    session_key = HKDF(shared_secret)

ciphertext = AESGCM_Encrypt(session_key, contract_bytes)
send_to_hr(ciphertext)

# Buyer signs
hash = SHA256(contract)
signature = ECDSA_Sign(buyer_priv, hash)
verify_signature(buyer_pub, signature)
```

5. Implementation Summary

The complete implementation is provided in `demo.py`. It utilises the Python `cryptography` library to realise ECC operations, AES-GCM encryption, and secure key storage.

The script performs the following steps:

1. Generates or loads ECDSA key pairs for H&R, SS, and Buyer.
2. Checks if a session key exists (`hr_ss_session.bin`); if not, performs an authenticated ECDH exchange.
3. Encrypts the contract using AES-GCM and sends it to H&R.
4. Buyer signs the contract digest + timestamp.
5. H&R verifies the signature and returns the signed contract to SS.

5.1 Output Files

- `contract_encrypted_to_hr.bin`
- `contract_signed_package_to_ss.bin`
- `contract.txt`

5.2 Evidence

Terminal logs and encrypted ciphertexts are stored in the appendix. Verification messages confirm successful signature validation and encryption operations.

6. Strengths and Limitations

6.1 Strengths

- **Confidentiality:** AES-GCM provides strong encryption.
- **Integrity:** GCM tag ensures tamper detection.
- **Authentication:** ECDSA validates identity of each party.
- **Forward Secrecy:** Ephemeral keys prevent compromise from revealing past messages.
- **Reusability:** Session key persistence simplifies repeated transactions.

6.2 Limitations

- **Key Management:** Requires careful storage and lifecycle rotation.
- **No Revocation Mechanism:** Protocol does not specify how compromised keys are revoked.
- **No Transport Layer:** The prototype lacks real network sockets; would rely on TLS in deployment.
- **Legal Integration:** Real-world digital signatures would require timestamping and audit trails under eIDAS compliance.

7. Discussion and Findings

The proposed design achieves secure document exchange for legal transactions in distributed settings. The integration of ECDH, AES-GCM, and ECDSA demonstrates end-to-end confidentiality and authentication. Repeated communication is made efficient via session key reuse, while first-time exchanges benefit from authenticated handshakes.

The implemented Python prototype validates the concept and produces verifiable artifacts. In practical deployment, integration with PKI, time-stamping authorities, and hardware-based secure elements (e.g., HSMs or TPMs) would further strengthen non-repudiation.

8. Conclusion

This report designed, justified, and implemented a secure protocol for digital contract exchange between solicitors and clients of Hackit & Run LLP. It balances cryptographic rigor with usability by supporting both first-time and repeated communication modes.

While the proof-of-concept code demonstrates local operations only, it forms a foundation for real-world deployment using secure channels (TLS), legal digital signature frameworks, and centralized certificate management. Future enhancements include integrating blockchain-based notarization and automatic key rotation schedules.

References

1. Dierks, T., & Rescorla, E. (2018). *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. IETF.

2. Krawczyk, H., & Eronen, P. (2010). *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. RFC 5869.

3. NIST. (2001). *Advanced Encryption Standard (AES)*. FIPS PUB 197.

4. National Cyber Security Centre (NCSC). (2023). *Using cryptography securely*. GOV.UK.

5. Ferguson, N., Schneier, B. (2003). *Practical Cryptography*. Wiley Publishing.

6. UK Government. (2020). *Electronic Identification and Trust Services for Electronic Transactions (eIDAS)*.

A. Appendix: Execution Evidence

Demo Output

```
                        > python -u "g:\My Drive\codes by art\HNR_assignment1\src\demo.py"
[demo] No session key found. Performing authenticated ECDH handshake (simulated).
[demo] Derived & stored session key (H&R <-> SS).
[demo] Seller -> H&R: Encrypted contract saved.
[demo] Buyer signature verified at H&R.
[demo] Signed package encrypted and saved for SS.
[demo] Demo complete. Files in: g:\My Drive\codes by art\HNR_assignment1\appendix
PS G:\My Drive\codes by art>
```

```
                        > python -u "g:\My Drive\codes by art\HNR_assignment1\src\demo.py"
[demo] Reusing existing session key (H&R <-> SS).
[demo] Seller -> H&R: Encrypted contract saved.
[demo] Buyer signature verified at H&R.
[demo] Signed package encrypted and saved for SS.
[demo] Demo complete. Files in: g:\My Drive\codes by art\HNR_assignment1\appendix
PS G:\My Drive\codes by art>
```

File Artifacts

- contract_encrypted_to_hr.bin
- contract_signed_package_to_ss.bin
- contract.txt

Notes

Private keys are not shared; only public keys and ciphertext are shown. Session keys are temporary and regenerated when deleted.