INTERNATIONAL JOURNAL FOR INNOVATIVE RESEARCH IN MULTIDISCIPLINARY FIELD
ISSN(O): 2455-0620 [Impact Factor: 9.47]
Monthly, Peer-Reviewed, Refereed, Indexed Journal with IC Value : 86.87
Volume - 11, Issue - 09, September - 2025

DOIs:10.2015/IJIRMF/202509031 --:-- Research Paper / Article / Review

Mathematical Applications using Hellman-David Cryptography

Khoulah Husayn Basheer Alghoul
Faculty of Education (Al Ajaylat)
Department of Mathematics, Sabratha University
Email - n.mbc2020@gmail.com

Abstract: *Many people mistakenly assume that Diffie–Hellman is an encryption algorithm, while in fact it is not. It is a key-agreement protocol designed specifically to establish a shared secret between two parties. This shared secret—essentially a large number—can later be used as a key to encrypt subsequent communications using a more efficient symmetric-key cipher such as AES or 3DES. Thus, Diffie–Hellman serves as a critical component in hybrid cryptosystems, where asymmetric cryptography solves the hard problem of key exchange, while the actual data encryption is performed by fast symmetric cryptography.*

One of the most important advantages of Diffie–Hellman is its ability to provide perfect forward secrecy (PFS)—a crucial property that ensures past communications remain confidential even if the server's long-term key is compromised in the future.

Key Words: *Diffie–Hellman, cryptography, symmetric encryption, asymmetric encryption.*

1. INTRODUCTION

The term cryptography is derived from the Greek words meaning "hidden writing." It is the science of concealing transmitted information so that it can be read only by the intended recipient.

The practice of cryptography dates back to antiquity; one of its earliest known examples is attributed to Julius Caesar. Modern cryptosystems are vastly more sophisticated, yet operate on similar principles. Most systems begin with an unencrypted message known as plaintext, which is then encrypted into an unreadable form known as ciphertext using one or more keys. The ciphertext is transmitted to the recipient. If intercepted while a strong algorithm is in use, ciphertext is useless to any unauthorized eavesdropper, who will be unable to decrypt it. The intended recipient, however, can easily decrypt the message—assuming possession of the correct decryption key. Cryptography has innumerable uses: from everyday end-to-end authentication of messages on WhatsApp and practical digital signatures on legal forms, to CPU-intensive cryptography used in cryptocurrency mining. Cryptography is a foundational pillar of the digital world and a core component of cybersecurity, protecting sensitive data from intruders and other cybercriminals.

2.Definition of Cryptography

Cryptography is one of the core operational techniques used in the field of information security, primarily helping to preserve the confidentiality of information.

It is the practice of protecting information using encryption algorithms, hash functions, and signatures. The information may be at rest (e.g., a file on a hard drive), in transit (e.g., electronic communications between two or more parties), or in use (e.g., during computation).

3.Objectives of Cryptography

- Confidentiality: Make information available only to authorized users.
- Integrity: Ensure information is not tampered with.

- Authentication: Confirm the authenticity of information or a user's identity.
- Non-repudiation: Prevent a user from denying prior commitments or actions.

Cryptography employs a number of low-level algorithms to achieve one or more of these security goals. These include encryption algorithms, digital-signature algorithms, hash algorithms, and other primitives. This paper outlines several of the most widely used low-level cryptographic algorithms (1).

Encryption Algorithms

An encryption algorithm transforms a plaintext message into encrypted information. Modern algorithms employ advanced mathematics and one or more keys. They make encryption relatively easy while making decryption practically impossible without knowledge of the key(s). Techniques are broadly divided into symmetric and asymmetric encryption, based on how their keys operate (2).

4.Classification of Cryptosystems

Cryptosystems can be classified along several axes:

1) Method of operation (substitution vs. transposition): All classical ciphers rely on substitution, transposition, or a combination. Substitution replaces each element of the plaintext (symbol, letter, group of bits, etc.) with a corresponding element. Transposition rearranges the plaintext elements without losing any element. Practical ciphers often employ multiple stages of substitution and transposition (3).

2) Number of keys used: If sender and receiver use the same key, the system is a symmetric-key (secret-key) cipher. If they use different keys, the system is asymmetric-key (public-key) cryptography. There are also functions that do not use a decryption key (e.g., cryptographic hash functions), where the output is a condensed "digest" of the original message.

3) Plaintext processing mode: Some algorithms process plaintext in blocks, yielding a block of ciphertext for each input block (block ciphers). Others process a continuous stream of input, encrypting one element at a time (stream ciphers). Issues related to direct digital notarization can be addressed using arbitrated signatures in various arrangements. This approach requires a trusted arbiter (third party) who formally authenticates the signed message, delays it, and forwards it to the recipient. The arbiter plays a critical role, and all parties must trust that the arbitral mechanism functions correctly. Arbitrated digital signatures are realized using public-key systems; the arbiter may or may not have access to the underlying message.

4.1 The Diffie–Hellman Method

Symmetric-key algorithms use the same key to encrypt plaintext and decrypt ciphertext. Symmetric encryption requires that all intended recipients have access to a shared secret key.

The Advanced Encryption Standard (AES) is one of the most well-known block ciphers, supporting 128-, 192-, or 256-bit keys. AES is often combined with Galois/Counter Mode (GCM)—AES-GCM—to provide an authenticated-encryption scheme (4).

AES is the global industry standard: its security is well-understood, and efficient software and hardware implementations are widely available.

4.2 How Symmetric Encryption Works

1. Key generation: The algorithm generates a random, secure encryption key.
2. Key exchange: The sender shares the encryption key securely with the receiver, often via a protected channel or coupled with an asymmetric method to ensure secrecy.
3. Data preparation: The algorithm prepares the plaintext (message/file) for encryption. Depending on the cipher, this may involve formatting the data into fixed-size blocks.
4. Encrypt: The cipher applies computations to the plaintext using the encryption key, producing ciphertext that appears random and is unreadable without the key.
5. Transmit/store ciphertext: The encrypted message is transmitted (for communications) or stored (for data at rest).
6. Decrypt: The receiver uses the same key to invert the process and recover the plaintext.

4.3 Common Uses of Symmetric Encryption

- Data storage: Encrypt files, databases, and backups.
- Secure communications: Protect data in transit (messages, email) against eavesdropping.
- Financial transactions: Secure sensitive information in online banking and payment systems.
- Device security: Encrypt data on personal devices and in enterprise environments to prevent unauthorized access.
- Network security: Protect network traffic (e.g., VPNs).

4.4 Risks of Symmetric Encryption

- Key management: Securely distributing and managing keys becomes increasingly difficult as the number of users/devices grows.
- Exposure from key compromise: If a symmetric key is compromised, all data encrypted with that key is exposed.
- No non-repudiation: Symmetric schemes cannot provide non-repudiation because both parties share the same key.
- Limited suitability over public networks: Because symmetric schemes require a pre-shared key, they are less suitable where an initial secure channel is unavailable.

By contrast, asymmetric cryptography uses mathematically related but distinct keys, enabling secure data exchange without sharing a private key. It is commonly used where secure key exchange and digital signatures are needed (HTTPS communications, email encryption, digital certificates).

5 Asymmetric (Public-Key) Cryptography

Public-key systems use pairs of related keys: a public key and a private key. Key pairs are generated using cryptographic algorithms based on hard mathematical problems known as one-way functions. Security depends on keeping the private key secret, while the public key can be widely distributed without compromising security. Public-key systems support several security goals, including digital signatures, Diffie–Hellman key exchange, key encapsulation, and public-key encryption.

Public-key algorithms underpin modern cryptographic security in applications and protocols that assure the confidentiality and authenticity of communications and data storage. They support many Internet standards, including TLS, SSH, S/MIME, and PGP. Compared to symmetric cryptography, public-key operations are often much slower, so real-world protocols combine them in hybrid cryptosystems (5).

When no secure channels exist—or when rotating keys frequently—public-key systems shine. Especially when messages must be kept confidential from other users, each potential pair of users would otherwise require a separate shared key in a purely symmetric setting. Publishing public keys openly avoids this burden, while only the private keys must be kept secret. Two famous public-key uses are digital signatures and public-key encryption:

1. Digital signatures: The sender uses a private key to create a signature over a message. Anyone with the corresponding public key can verify the signature matches the message.
2. Public-key encryption: Anyone with the public key can encrypt a message into ciphertext, but only the holder of the corresponding private key can decrypt it.

6 Encryption Key Management

Key management encompasses generating, exchanging, and administering cryptographic keys to keep encrypted data secure.

To avoid losing access or exposing data, organizations invest in key-management systems. Common features include central consoles for managing encryption and key policies, encryption of local and cloud-hosted data, role/group-based access controls and audit logging, automated key-lifecycle operations, and integration with emerging technologies.

A sufficiently large (usually random) number is used to seed the generation of an acceptable key pair for an asymmetric algorithm.

Example: A message is digitally signed with Alice’s private key, but the message itself is not encrypted. Using Alice’s public key, Bob verifies authenticity.

Cryptographic key management is the process of generating, exchanging, and administering encryption keys to ensure the security of encrypted data.

Think of encryption as a safe: if you forget the secret code or it falls into the wrong hands, you risk losing access to your valuables—or having them stolen. Similarly, if institutions do not properly manage their encryption keys, they may lose access to encrypted data or expose themselves to data-security breaches (6).

To avoid such attacks, organizations often invest in key management systems. These services are crucial, because organizations frequently manage a complex web of encryption keys, and many threat actors know where to look for them. Key-management solutions often include features such as:

- A centralized console for managing encryption, key policies, and configurations
- Encryption of on-premises and cloud-hosted data at the file, database, and application levels
- Role- and group-based access controls and audit logging to help address compliance requirements
- Automated key-lifecycle operations
- Integration with the latest technologies, such as artificial intelligence, to improve key management using analytics and automation

An unpredictable number (typically large and random) is used to initiate the generation of an acceptable key pair suitable for use by an asymmetric-key algorithm. (7)

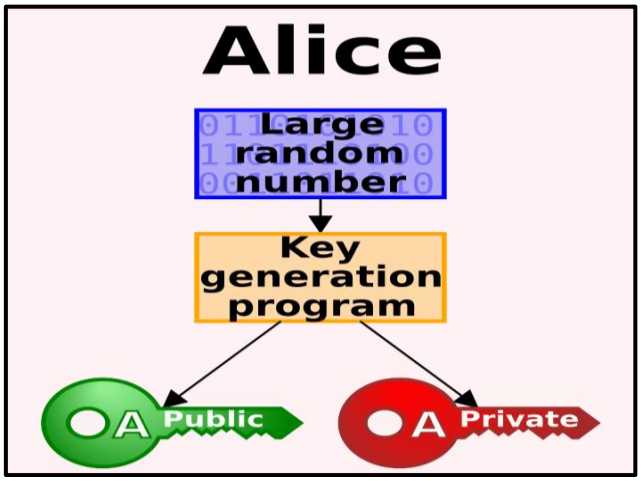

Figure (1): Public-key encryption.

7 Asymmetric-key algorithm.

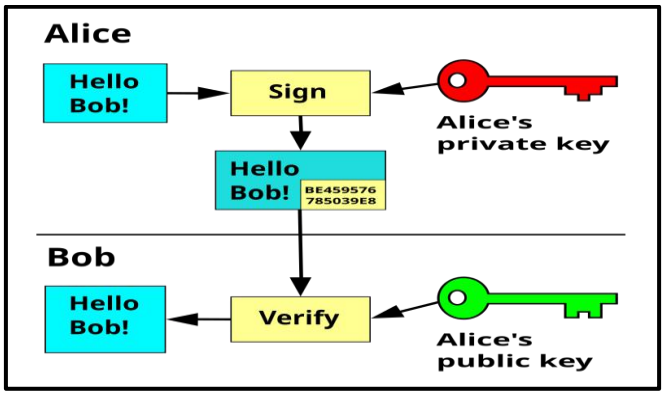

Figure (2): Generating an acceptable key pair suitable for use by an asymmetric-key algorithm.

Example: The message is digitally signed using Alice's private key, but the message itself is not encrypted.

Alice signs the message with her private key.

Using Alice's public key, Bob can verify that Alice sent the message and that the message has not been modified. (7)

7.3 Asymmetric-key encryption

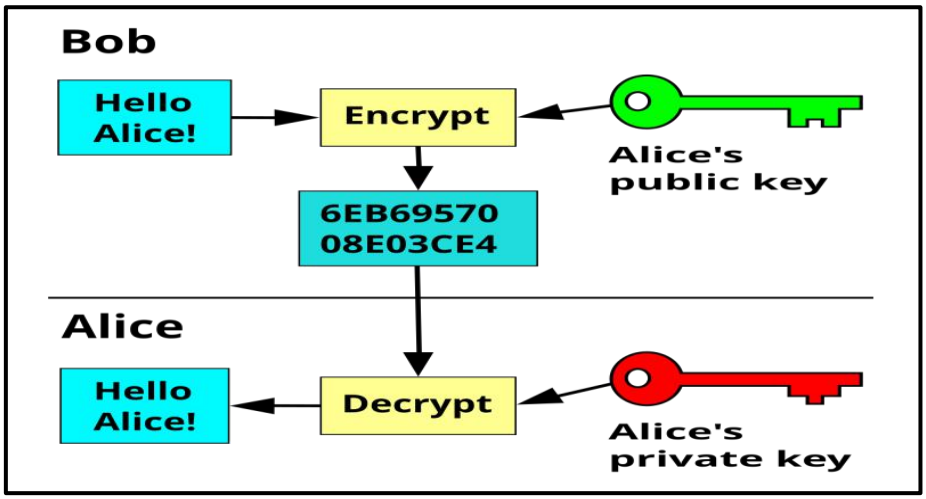

Figure (3): Asymmetric-key encryption system.

In an asymmetric-key encryption system, anyone can encrypt messages using a public key, but only the holder of the corresponding private key can decrypt that message. The security of the system depends on the secrecy of the private key, which must not become known to anyone else. (7)

8. Diffie–Hellman Key Exchange

In the Diffie–Hellman key-exchange scheme, each party generates a public/private key pair and distributes the public key. After obtaining an authentic copy (this is critical), the parties compute a shared secret.

Diffie–Hellman key exchange is fundamental to key management. It enables two parties to exchange encryption keys securely over public channels and establish a shared secret for subsequent secure communication.

Security rests on the hardness of the discrete logarithm problem (DLP). Diffie–Hellman appears in protocols such as SSL/TLS. WhatsApp employs Diffie–Hellman as part of the Signal Protocol to provide end-to-end encryption. Diffie–Hellman is also widely used in VPNs and secure email systems (8).

Symmetric vs. Asymmetric Encryption

Unlike symmetric encryption—where the same key is shared—public and private keys in asymmetric cryptography are mathematically related but not identical, allowing secure data exchange without sharing the private key (9). Asymmetric schemes are used where secure key exchange and digital signatures are essential (HTTPS, email encryption, digital certificates).

Symmetric Keys vs. Asymmetric Keys

- Symmetric keys: A single shared key is used for both encryption and decryption; efficient for large data but challenging to share securely.
- Asymmetric keys: Keys come in pairs; the public key encrypts while the private key decrypts, enabling secure key exchange without pre-shared secrets.

9. Number Theory Foundations for Diffie–Hellman

Modular Arithmetic and Exponentiation

Diffie–Hellman relies on number-theoretic principles, especially modular arithmetic, where calculations are performed within a fixed range and results are taken modulo a number called the modulus. For example, 13 mod 5 = 3 because 13 = 5 × 2 + 3. Modular exponentiation plays a central role in the protocol.

The Discrete Logarithm Problem (DLP)

Diffie–Hellman's security rests on the computational hardness of the discrete logarithm problem—a trapdoor function: easy to compute in one direction, intractable to invert. Formally, given a large prime p, a primitive root g, and a value h, find x satisfying g^x ≡ h (mod p). Computing h from g and x is efficient; inverting to recover x is extremely difficult when parameters are large.

Relationship Between DLP and the Diffie–Hellman Problem (DHP)

In DHP, an attacker who observes public values A ≡ g^a (mod p) and B ≡ g^b (mod p) finds it hard to compute the shared secret g^(ab) (mod p) directly. Anyone who can solve DLP can trivially solve DHP by recovering a from A and then computing S = B^a (mod p). The converse may not hold; DHP might be harder than DLP.

10. Protocol Steps and Worked Example

Methodical Steps

1. Agree on public parameters: choose a large prime p and a generator g (public).
2. Choose private keys: each party selects a random secret integer (a, b).
3. Compute public keys: A = g^a mod p; B = g^b mod p.
4. Exchange public keys: Alice and Bob exchange A and B over an insecure channel.
5. Compute the shared secret: Alice computes S = B^a mod p; Bob computes S = A^b modp.

Illustrative Example

Public parameters: p = 23, g = 5.
Private keys: Alice chooses a = 4; Bob chooses b = 3.
Public keys: A = 5^4 mod 23 = 4; B = 5^3 mod 23 = 10.
Exchange: Alice sends A = 4; Bob sends B = 10.
Shared secret: Alice computes 10^4 mod 23 = 18; Bob computes 4^3 mod 23 = 18.

11. Security Analysis

Diffie–Hellman's security depends on the absence of efficient algorithms for the discrete logarithm over large parameters. While advanced algorithms such as the Number Field Sieve exist, their resource requirements become impractical at the sizes used today.

Inherited Weakness: Man-in-the-Middle (MitM)

In its basic form, Diffie–Hellman does not authenticate the parties, leaving it vulnerable to a MitM attack. An attacker can intercept and replace public keys during the exchange, establishing separate shared secrets with each party and relaying messages between them. In practice, Diffie–Hellman is paired with authentication (digital signatures, certificates) to prevent MitM.

Diffie–Hellman vs. RSA

Aspect	Diffie–Hellman (DH)	RSA
Purpose	Key-agreement protocol only	Encryption & digital signatures
Math foundation	Hardness of discrete logarithm	Hardness of integer factorization

Perfect Forward Secrecy	Supported in ephemeral modes (DHE)	Not inherent by default
Authentication	Not provided by DH alone	Commonly used via certificates (signing)

Traditional DH vs. Elliptic-Curve DH (ECDH)

Aspect	**Traditional DH**	**ECDH**
Math foundation	Discrete logarithm (DLP)	Discrete log on elliptic curves (ECDLP)
Key sizes	Large (2048–4096 bits)	Smaller (≈256–384 bits) for comparable security
Efficiency	Lower (heavier compute/bandwidth)	Higher—ideal for constrained devices

Practical Applications in Secure Communications

- TLS/HTTPS: DH establishes a pre-master secret; ephemeral DH (DHE) provides PFS.
- SSH: DH establishes a shared session key; the exchange is signed by the server's host key.
- VPNs: DH during IKE creates encrypted tunnels; modern deployments prefer large or elliptic-curve groups.

Software Applications Using Diffie–Hellman

Examples include secure messaging (e.g., Signal/WhatsApp), VPN suites, and secure email systems. Some educational examples demonstrate implementing Diffie–Hellman to protect student data :

```
# -*- coding: utf-8 -*-

import json, base64, secrets

from dataclasses import dataclass


P_HEX = """

FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1

29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD

EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245

E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED

EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D

C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F

83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D

670C354E 4ABC9804 F1746C08 CA237327 FFFFFFFF FFFFFFFF

""".replace(" ", "").replace("\n", "")

P = int(P_HEX, 16)

G = 2
```

```
# --- HKDF + AES-GCM ---
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
def _int_to_bytes(x: int) -> bytes:
    return x.to_bytes((x.bit_length() + 7) // 8, "big")
def dh_generate_keypair():
    private = secrets.randbelow(P - 2) + 2  # 2..p-1
    public = pow(G, private, P)
    return private, public
def dh_shared_secret(my_private: int, their_public: int) -> int:
    if not (2 <= their_public <= P - 2):
        raise ValueError("invalid public key.")
    return pow(their_public, my_private, P)
def derive_aes_key(shared_secret_int: int, salt: bytes, info: bytes = b"student-data-v1", length=32) -> bytes:

    shared_bytes = _int_to_bytes(shared_secret_int)
    hkdf = HKDF(algorithm=hashes.SHA256(), length=length, salt=salt, info=info)
    return hkdf.derive(shared_bytes)

def aesgcm_encrypt(key: bytes, plaintext: bytes, aad: bytes = None):
    aesgcm = AESGCM(key)
    nonce = secrets.token_bytes(12)
    ciphertext = aesgcm.encrypt(nonce, plaintext, aad)
    return nonce, ciphertext

def aesgcm_decrypt(key: bytes, nonce: bytes, ciphertext: bytes, aad: bytes = None) -> bytes:
    aesgcm = AESGCM(key)
    return aesgcm.decrypt(nonce, ciphertext, aad)
```

```
@dataclass
class Student:
    student_id: str
    name: str
    major: str
    gpa: float

    def to_json_bytes(self) -> bytes:

        return json.dumps(self.__dict__, ensure_ascii=False, separators=(",", ":")).encode("utf-8")
def main():
    a_priv, a_pub = dh_generate_keypair()  # Alice
    b_priv, b_pub = dh_generate_keypair()  # Bob
    s_alice = dh_shared_secret(a_priv, b_pub)
    s_bob   = dh_shared_secret(b_priv, a_pub)

    assert s_alice == s_bob,
    shared_secret = s_alice

    salt = secrets.token_bytes(16)
    key = derive_aes_key(shared_secret, salt)

    student = Student(
        student_id="2025001",
        name="Ahmed Ali",
        major="Computer Science",
        gpa=3.41,
    )
    plaintext = student.to_json_bytes()
```

```
aad = b"university-records-v1"
nonce, ciphertext = aesgcm_encrypt(key, plaintext, aad)
package = {
    "dh_params": {
        "p_hex": P_HEX,
        "g": G,
        "a_pub": format(a_pub, "x"),
        "b_pub": format(b_pub, "x"),
    },
    "kdf": {
        "salt_b64": base64.b64encode(salt).decode(),
        "info": "student-data-v1",
    },
    "aead": {
        "nonce_b64": base64.b64encode(nonce).decode(),
        "ciphertext_b64": base64.b64encode(ciphertext).decode(),  # ct || tag
        "aad_b64": base64.b64encode(aad).decode(),
        "scheme": "AES-256-GCM",
    },
}

print(json.dumps(package, ensure_ascii=False, indent=2))
rec_salt = base64.b64decode(package["kdf"]["salt_b64"])
rec_nonce = base64.b64decode(package["aead"]["nonce_b64"])
rec_ct = base64.b64decode(package["aead"]["ciphertext_b64"])
rec_aad = base64.b64decode(package["aead"]["aad_b64"])

rec_key = derive_aes_key(shared_secret, rec_salt)
recovered = aesgcm_decrypt(rec_key, rec_nonce, rec_ct, rec_aad)
```

```
    print(recovered.decode("utf-8"))


if __name__ == "__main__":
    main()
```

12. Conclusion

The Diffie–Hellman key-exchange method is a cornerstone of modern cryptography, providing a practical solution to the challenge of exchanging keys over insecure channels. Its security is grounded in the hardness of the discrete logarithm problem, and it underpins widely used protocols such as TLS, SSH, and VPNs. Although basic DH lacks authentication and is susceptible to MitM, combining it with digital signatures and related mechanisms yields robust and flexible security systems.

A key advantage of DH is its support for perfect forward secrecy, ensuring past sessions remain confidential even if a long-term key is later compromised. Looking ahead, quantum computing may threaten these hardness assumptions, motivating research into post-quantum cryptography.

References:

1. Gollmann, Dieter (2011). Computer Security (2nd ed.). John Wiley & Sons, Ltd. ISBN 978-0470741153.
2. Diffie, Whitfield. “The First Ten Years of Public-Key Cryptography.” Proceedings of the IEEE, vol. 76, no. 5, May 1988, pp. 560–577.
3. Menezes, Alfred; van Oorschot, Paul; Vanstone, Scott (1997). Handbook of Applied Cryptography. CRC Press.
4. Singh, Simon (1999). The Code Book: The Evolution of Secrecy from Mary Queen of Scots to Quantum Cryptography. Doubleday.
5. Hellman, Martin E. “An Overview of Public Key Cryptography.” IEEE Communications Magazine, May 2002, pp. 42–49.
6. Buchanan, Bill. “Diffie–Hellman Example in ASP.NET.” (retrieved 2015-08-27).
7. Buchmann, Johannes A. (2013). Introduction to Cryptography (2nd ed.). Springer Science+Business Media.
8. EUROCRYPT 2014. A Heuristic Quasi-Polynomial Algorithm for Discrete Logarithm in Finite Fields of Small Characteristic. LNCS 8441.
9. RFC 4306. Internet Key Exchange (IKEv2) Protocol. IETF.