

Forum:[Hack Forums](#)

Thread: [Creating a Dynamic XOR Cipher: HMAC-Seeded Security \(Custom XOR encryption\)\)](#)

darkterror - February 2, 2025, 1:04 pm

Hi guys,

Today i am gonna share my custom "pseudo polymorphic" XOR encryption for files. I am still kinda new to whole cryptography topic, so i am open to discussion about possible improvements or hints how to make it better.

1. Core Mechanism:

It's a stream cipher that uses XOR but with a twist: the XOR key changes dynamically for every byte, based on a seed derived from your password and the data itself.

Think of it as a "chameleon" XOR—the key evolves as it encrypts/decrypts.

2. Key Steps:

Seed Generation:

Uses HMAC-SHA256 to create a unique 32-bit seed from your key + a fixed message ("PolymorphicXorSeed"). This ties the seed to your key securely.

Dynamic Key per Byte:

For each byte, the XOR key is built by:

- Rotating the seed (left/right, based on byte index).
- XORing the rotated value with a byte from your key (reused cyclically).

Seed Evolution:

After encrypting each byte, the seed is rotated right by 2 bits and mixed with the plaintext byte. This creates a feedback loop—every byte alters the seed, making the next key unpredictable.

3. Decryption Mirroring:

Decryption reverses the process by recomputing the same dynamic keys, thanks to the seed's dependence on plaintext bytes (which are revealed during decryption).

4. Secure Key Generation:

Uses RandomNumberGenerator (cryptographically secure RNG) to create random keys, avoiding weak passwords.

Benefits Over Conventional XOR:

? Dynamic Keys:

Conventional XOR: Uses a fixed key (e.g., key = 0x3A, 0xAA...). Same plaintext = same ciphertext. Easy to crack with patterns.

This Code: Every byte's XOR key depends on the evolving seed + index. Even identical plaintext bytes get different ciphertext!

? Key Sensitivity:

A 1-bit change in the plaintext or key alters the seed cascade, radically changing all subsequent ciphertext bytes.

? Seed Entropy:

HMAC-SHA256 ensures the seed isn't guessable from the key alone. Conventional XOR often uses trivial key-to-seed conversions.

? Resilience to Repetition Attacks:

In conventional XOR, reused keys expose patterns (e.g., $\text{plaintext1} \oplus \text{ciphertext1} = \text{key}$). Here, the dynamic keys break this relationship.

? Not "Unbreakable":

Still lacks modern cipher features like authentication (tamper detection) or an IV (to prevent same-input/same-output). But it's way stronger than basic XOR.

```
Code using System;  
using System.IO;  
using System.Text;  
using System.Security.Cryptography;
```

```
namespace XORpolymorphicFile  
{  
    public static class PolymorphicXorCipher  
    {  
        public static byte[] Encrypt(byte[] plainData, byte[] key)  
        {  
            if (plainData == null)  
                throw new ArgumentNullException(nameof(plainData));  
            if (key == null || key.Length == 0)  
                throw new ArgumentException("Key cannot be null or empty.", nameof(key));  
  
            byte[] output = new byte[plainData.Length];  
            uint seed = DeriveSeed(key);  
  
            for (int i = 0; i < plainData.Length; i++)  
            {  
                byte dynamicKey = ComputeDynamicKey(seed, i, key);  
                byte plainByte = plainData[i];  
                byte cipherByte = (byte)(plainByte ^ dynamicKey);  
                output[i] = cipherByte;  
                seed = UpdateSeedPlain(seed, plainByte);  
            }  
        }  
    }  
}
```

```

    return output;
}

public static byte[] Decrypt(byte[] cipherData, byte[] key)
{
    if (cipherData == null)
        throw new ArgumentNullException(nameof(cipherData));
    if (key == null || key.Length == 0)
        throw new ArgumentException("Key cannot be null or empty.", nameof(key));

    byte[] output = new byte[cipherData.Length];
    uint seed = DeriveSeed(key);

    for (int i = 0; i < cipherData.Length; i++)
    {
        byte dynamicKey = ComputeDynamicKey(seed, i, key);
        byte cipherByte = cipherData[i];
        byte plainByte = (byte)(cipherByte ^ dynamicKey);
        output[i] = plainByte;
        seed = UpdateSeedPlain(seed, plainByte);
    }

    return output;
}

private static uint DeriveSeed(byte[] key)
{
    byte[] seedMessage = Encoding.UTF8.GetBytes("PolymorphicXorSeed");//You can set your
own seed
    using (var hmac = new HMACSHA256(key))
    {
        {
            byte[] hash = hmac.ComputeHash(seedMessage);
            return BitConverter.ToUInt32(hash, 0);
        }
    }
}

private static byte ComputeDynamicKey(uint seed, int index, byte[] key)
{
    byte keyByte = key[index % key.Length];
    byte rotated;

    switch (index % 3)
    {
        {
            case 0:
                rotated = (byte)(RotateLeft(seed, 3) & 0xFF);
                break;
            case 1:
                rotated = (byte)(RotateRight(seed, 5) & 0xFF);
                break;
            default:
                rotated = (byte)(RotateLeft(seed, 7) & 0xFF);
                break;
        }
    }
}

```

```

    }

    return (byte)(rotated ^ keyByte);
}

private static uint UpdateSeedPlain(uint seed, byte plainByte)
{
    return RotateRight(seed, 2) ^ plainByte;
}

private static uint RotateLeft(uint value, int bits)
{
    return (value << bits) | (value >> (32 - bits));
}

private static uint RotateRight(uint value, int bits)
{
    return (value >> bits) | (value << (32 - bits));
}

public static byte[] GenerateRandomKey(int keyLength)
{
    byte[] key = new byte[keyLength];
    using (var rng = RandomNumberGenerator.Create())
    {
        rng.GetBytes(key);
    }
    return key;
}

}

class Program
{
    static void Main()
    {
        string inputFilePath = @"<PATH TO YOUR EXECUTABLE>";
        string encryptedFilePath = "Encrypted.exe";
        string decryptedFilePath = "decrypted.exe";
        byte[] key = PolymorphicXorCipher.GenerateRandomKey(16);
        Console.WriteLine("Random key (base64): " + Convert.ToBase64String(key));

        try
        {
            byte[] originalBytes = File.ReadAllBytes(inputFilePath);
            Console.WriteLine("Original file read: " + inputFilePath);
            byte[] encryptedBytes = PolymorphicXorCipher.Encrypt(originalBytes, key);
            File.WriteAllBytes(encryptedFilePath, encryptedBytes);
            Console.WriteLine("Encrypted file saved as: " + encryptedFilePath);
            byte[] encryptedFileBytes = File.ReadAllBytes(encryptedFilePath);
            byte[] decryptedBytes = PolymorphicXorCipher.Decrypt(encryptedFileBytes, key);
            File.WriteAllBytes(decryptedFilePath, decryptedBytes);
            Console.WriteLine("Decrypted file saved as: " + decryptedFilePath);
        }
    }
}

```

```

        catch (Exception ex)
        {
            Console.WriteLine("Error: " + ex.Message);
        }

        Console.WriteLine("Press any key to exit . . .");
        Console.ReadKey();
    }
}
}

```

In summary, my custom encryption uses a dynamically evolving key stream (driven by a robust seed derived via HMAC and continuously updated) to achieve a pseudo-polymorphic behavior. This makes it more secure and less predictable than traditional XOR encryption, though it's still a toy and proof-of-concept example compared to modern ciphers. Feel free to discuss or suggest improvements—I'm all ears!

derfdo716 - March 1, 2025, 12:31 pm

Brother, I have some advice for this one:

In ComputeDynamicKey, if seed is 0, every case of the switch statement will set rotated = 0; Problem is that the switch statement could also have a result of 0 if only the first or last N bits are set as the AND 0xFF dropping bits.

Also, because of the AND 0xFF operation dropping bits following the ROT, some bits in the ciphertext will be equal to what you would get from XOR operation of the raw stream.

darkterror - April 2, 2025, 11:03 am

(Mar 1, 2025 01:31 PM)derfdo716 Wrote: [CITE](#) Brother, I have some advice for this one:

In ComputeDynamicKey, if seed is 0, every case of the switch statement will set rotated = 0; Problem is that the switch statement could also have a result of 0 if only the first or last N bits are set as the AND 0xFF dropping bits.

Also, because of the AND 0xFF operation dropping bits following the ROT, some bits in the ciphertext will be equal to what you would get from XOR operation of the raw stream.

i am working on new "stronger" encryption, good point will review some of the logic

eagle8900 - April 12, 2025, 8:36 am

i need free keylogger pls

cruzpassport1 - August 14, 2026, 11:50 pm

the password to the file is not correct

cruzpassport1 - August 23, 2026, 8:49 pm

he password to the file is not correct