

Sprzedaż licencji z Paddle

Programowanie daje ogromną satysfakcję i dla niejednego z nas stanowi po prostu życiową pasję. Niestety jednak, z samej satysfakcji trudno jest wyżyć i od czasu do czasu efekty naszej pracy trzeba po prostu sprzedać. W niniejszym artykule pokażę, jak wprowadzić do aplikacji system licencji oraz jak zintegrować go z mechanizmem płatności Paddle.

I PODSTAWY

Istnieje cały szereg mechanizmów sprzedaży oprogramowania. Bardzo popularny stał się ostatnimi laty system subskrypcyjny, ale unosi się wokół niego duża aura niechęci, bo coraz częściej zdarza się, że niektóre duże korporacje (*khy khy*Adobe*khy khy*) zaczęły w ten sposób sprzedawać oprogramowanie, które w swojej istocie wcale nie jest usługą. Wciąż więc bardziej lubianym, przez co otwierającym aplikację na szersze audytorium jest po prostu mechanizm licencji.

Działa on stosunkowo prosto. Użytkownik ściąga aplikację, działającą w trybie ewaluacyjnym: niektóre funkcje działają w sposób ograniczony lub nie działają wcale. Użytkownik ma jednak możliwość zapoznać się z oprogramowaniem i ocenić, czy spełnia ono jego oczekiwania. Jeżeli oceni, że tak jest, może przejść do sklepu, zapłacić, a wówczas na swój adres e-mail otrzymuje klucz licencyjny, którego wpisanie do aplikacji odblokowuje jej pełną funkcjonalność.

Opiszę dziś, jak wygląda implementacja takiego mechanizmu licencjonowania aplikacji oraz jak zintegrować ten mechanizm z dostawcą usług płatności – w naszym przypadku będzie to stosunkowo popularny Paddle.

I KONSTRUKCJA LICENCJI

Najprostszą wersją mechanizmu licencyjnego jest klucz licencyjny. Za pomocą pewnego algorytmu generujemy ciąg liter i cyfr; przekazujemy go użytkownikowi, a w aplikacji tym samym algorytmem weryfikujemy, czy jest on prawidłowy. Pewną wariacją na temat jest użycie jako klucza licencyjnego pary: nazwy użytkownika (lub adresu e-mail) oraz klucza, który generowany jest na podstawie tego adresu tym samym algorytmem zarówno podczas generowania klucza podczas sprzedaży, jak i podczas jego późniejszej weryfikacji. Praktyka pokazuje jednak, że tego typu klucze są zwykle stosunkowo łatwo łamane i dlatego lepiej podejść do problemu nieco inaczej.

Znacznie prostszym do oprogramowania, a jednocześnie trudnym do złamania mechanizmem jest wykorzystanie koncepcji, którą znamy ze środowiska webowego, a która obecna jest tam w formie tokenów. Licencję zapisujemy jako plik tekstowy – na przykład w formacie JSON – przechowując w nim adres e-mail użytkownika, a także wszystkie inne konieczne informacje, na przykład datę aktywacji licencji lub jej wersję (pro, ultimate etc.). Następnie obliczamy skrót tak przygotowanej licencji, a potem szyfrujemy go za pomocą silnego algorytmu kryptograficznego. Dla wygody, całość (licencję i podpis cyfrowy) możemy przekształcić na Base64, a potem dostarczyć użytkownikowi.

Kiedy użytkownik wprowadzi taki klucz do aplikacji (z uwagi na rozmiar można go też przekazać w formie niewielkiego pliku teksto-

wego), obliczamy skrót ponownie, rozszyfrowujemy obecny w pliku licencyjnym podpis cyfrowy za pomocą klucza publicznego przechowywanego w aplikacji i na końcu porównujemy oba skróty. Jeżeli są równe – oznacza to, że dane zawarte w licencji są oryginalne. Jeśli się różnią, oznacza to, że ktoś musiał majstrować przy licencji, którą należy wtedy odrzucić.

Jeżeli skorzystamy z algorytmów uznawanych za bezpieczne (oraz ich sprawdzonych implementacji) oraz użyjemy odpowiednio długiego klucza, złamanie licencji będzie równoważne złamaniu algorytmu szyfrującego, co – jak możemy bezpiecznie założyć – jest raczej niemożliwe.

I KLASA SERWISU LICENCYJNEGO

W pierwszym etapie przygotujemy klasę, która dostarczy nam wszystkie funkcje związane z obsługą licencji.

Przed wszystkim będziemy potrzebować kluczy RSA, które będziemy wykorzystywać podczas generowania licencji oraz ich weryfikacji. Obecnie za bezpieczne uważane są klucze 2048-bitowe, natomiast zalecane – jako „odporne na przyszłość” – są 3072- lub 4096-bitowe. Na potrzeby przykładu skorzystamy z klucza 2048-bitowego.

Konieczne będzie oczywiście również napisanie dwóch metod, które – odpowiednio – wygenerują i zweryfikują licencję. Do tego będziemy naturalnie potrzebować modelu reprezentującego surowe dane, które chcemy zaszyfować w licencji. Od niego właśnie zaczniemy naszą implementację.

Listing 1. Klasa reprezentująca dane zawarte w licencji

```
public class LicenseData
{
    public string Email { get; set; } = string.Empty;

    public string ToBase64()
    {
        return Convert.ToBase64String(ToBytes());
    }

    public static LicenseData FromBase64(string base64)
    {
        return FromBytes(Convert.FromBase64String(base64));
    }

    internal byte[] ToBytes()
    {
        var values = new string[] { Email };
        var json = JsonSerializer.Serialize(values);
        return Encoding.UTF8.GetBytes(json);
    }

    internal static LicenseData FromBytes(byte[] data)
```

```

{
    try
    {
        var json = Encoding.UTF8.GetString(data);
        var values = JsonSerializer
            .Deserialize<string[]>(json);

        if (values == null || values.Length < 1)
            throw new FormatException(
                "Invalid license data format.");

        return new LicenseData
        {
            Email = values[0]
        };
    }
    catch (JsonException ex)
    {
        throw new FormatException(
            "Invalid license data format.", ex);
    }
}

```

Na potrzeby przykładu możemy przyjąć, że licencja będzie zawierała tylko adres e-mail. Nic nie stoi na przeszkodzie, aby zapisywać w niej więcej informacji; opisany powyżej mechanizm jest też kompatybilny wstecz, o ile będziemy tylko dodawać nowe dane. Może dziwić sposób, w jaki serializujemy dane do JSON – poprzez zapisanie ich w tablicy. Dzieje się tak po to, aby zserializowane dane zajmowały możliwie jak najmniej miejsca (a pamiętajmy, że dane zapisane w Base64 zajmują więcej miejsca niż oryginał). W naszym przypadku klucz licencyjny będzie przesyłany do użytkownika w treści wiadomości e-mail; jeżeli przesadzimy z ilością danych, postawimy go przed ścianą tekstu, który będzie musiał przekopiować do aplikacji. Wtedy klucz licencyjny trzeba byłoby już zawrzeć w pliku.

Teraz możemy zaimplementować klasę, która wygeneruje klucze RSA oraz pozwoli generować i weryfikować licencje.

Listing 2. Serwis licencyjny

```

internal class LicenseService : ILicenseService
{
    private const int KEY_SIZE = 2048;

    private static readonly HashAlgorithmName ▼
        HASH_ALGORITHM = HashAlgorithmName.SHA256;
    private static readonly RSASignaturePadding ▼
        SIGNATURE_PADDING = RSASignaturePadding.Pss;

    public void GenerateKeys(
        Stream bothKeysStream,
        Stream publicKeyStream)
    {
        using (var rsa = RSA.Create(KEY_SIZE))
        {
            byte[] privateKeyBytes = rsa.ExportRSAPrivateKey();
            bothKeysStream.Write(
                privateKeyBytes,
                0,
                privateKeyBytes.Length);

            byte[] publicKeyBytes = rsa.ExportRSAPublicKey();
            publicKeyStream.Write(
                publicKeyBytes,
                0,

```

```

                publicKeyBytes.Length);
        }
    }

    public string GenerateLicenseKey(
        LicenseData data,
        Stream privateKeyStream)
    {
        using (var rsa = RSA.Create())
        {
            byte[] keyBytes = ReadAllBytes(
                privateKeyStream);
            rsa.ImportRSAPrivateKey(keyBytes, out _);

            byte[] dataBytes = data.ToBytes();
            byte[] signature = rsa.SignData(
                dataBytes,
                HASH_ALGORITHM,
                SIGNATURE_PADDING);

            using (var ms = new MemoryStream())
            using (var writer = new BinaryWriter(ms))
            {
                writer.Write(dataBytes.Length);
                writer.Write(dataBytes);
                writer.Write(signature);

                return Convert.ToBase64String(
                    ms.ToArray());
            }
        }
    }

    public LicenseData? VerifyLicenseKey(
        string licenseKey,
        Stream publicKeyStream)
    {
        try
        {
            byte[] licenseBytes =
                Convert.FromBase64String(licenseKey);

            if (licenseBytes.Length < sizeof(int) + 1)
                return null;

            using (var ms = new MemoryStream(licenseBytes))
            using (var reader = new BinaryReader(ms))
            {
                int dataLength = reader.ReadInt32();
                if (dataLength <= 0 ||
                    dataLength >
                        licenseBytes.Length - sizeof(int) - 1)
                    return null;

                byte[] dataBytes =
                    reader.ReadBytes(dataLength);
                byte[] signature =
                    reader.ReadBytes(
                        (int)(ms.Length - ms.Position));

                if (signature.Length == 0)
                    return null;

                using (var rsa = RSA.Create())
                {
                    byte[] keyBytes =
                        ReadAllBytes(publicKeyStream);
                    rsa.ImportRSAPublicKey(keyBytes, out _);

                    if (rsa.VerifyData(
                        dataBytes,
                        signature,

```