データ特性に最適化したHTML5アプリケーションのパッケージ方式

矢崎 孝一† 伊藤 栄信† 二村 和明†

†株式会社富士通研究所
211-8588 神奈川県川崎市中原区上小田中 4-1-1
{Yasaki.kouichi, itou.hidenobu, kazuaki.nimura}@jp.fujitsu.com

あらまし アプリケーション記述言語として、マルチプラットフォームで動作し柔軟な処理記述ができるHTML5が着目されている。一般的に、HTML5で作成したアプリケーションを端末に配信する際にパッケージングを行い、情報漏洩を防ぐために暗号化が行われる。しかし、パッケージを暗号化したままアプリケーションを読み出すと、暗号処理による読出負荷が高く、体感速度が低下する問題がある。そこで、暗号化による読出負荷を軽減するため、パッケージ復号部の高速化と、その復号部へのパッケージ最適化を行う手法について提案を行う。これにより、パッケージ全体を暗号化した場合においても、平文と比較して約10%の負荷増で読み出しが可能となった。

Proposal of a package scheme of HTML5 application by optimizing based on data characteristics

Kouichi Yasaki† Hidenobu Ito† Kazuaki Nimura†

†Fujitsu Laboratories LTD.
1-1, Kamikodanaka 4-chome, Nakahara, Kawasaki
211-8588, JAPAN
{Yasaki.kouichi, itou.hidenobu, kazuaki.nimura }@jp.fujitsu.com

Abstract HTML5 has "write once, run anywhere" characteristic, and abundant APIs for rich UI realizing various services. Generally, an application programmed by HTML5 is packaged, and delivered to mobile terminal. The package should be encrypted in order to prevent information leakage; however it raises a problem that the load of reading data from the package is heavier. Then we propose a packaging method to improve the load of reading by optimizing each encryption/compression algorithm in the package based on a decryption ability of mobile terminal, and a characteristics of accessing data from browser. We implemented the proposal and confirmed that the load of reading data became about 10% increase compared to the plain text.

1 はじめに

スマートフォンや、タブレットなどのモバイル端末の普及が進んでいる。そのアプリケーションを記述する言語として、HTML5 が注目を集めている。

HTML は、文章と共にその構造や見栄えに関する指定をタグとしてテキストファイルに記述するマークアップ言語であり、W3C(World Wide Web Consortium)で標準化が進められ

ている。HTML5 では、動的なウェブページを作成するための API や、データ保存用の API が追加されることにより、オンライン・オフラインにかかわらず、利用者の操作に応じてダイナミックに表示をかえるアプリケーションを記述することができるようになった。また、HTML5 は、標準化団体が定めた API であるため、その API を使用することで特定 OS への依存性がないアプリケーションを作ることができる。

この HTML5 で作成したアプリケーションを端末に配信する際にはパッケージングが行われる。その際、情報漏洩を防ぐ目的で暗号化が行われる場合がある。暗号化をした場合、アプリケーション実行のどこかのタイミングで復号処理が行われる。

この復号処理を、アプリケーション実行と同時に行うとパッケージからの読み出しに時間がかかり、アプリケーションの体感速度が低下する。それより前のタイミングで復号を行うと、その間、情報漏洩のリスクが高まることになる。

そこで本稿では、アプリケーション実行時に復号する場合であっても、データ読み出し速度が低下しないようにするため、データ特性に応じたパッケージ最適化と、復号機能の最適化による改善手法の提案を行う。

以下、2 章では関連研究として、HTML5 アプリケーションのパッケージング手法の動向に触れ、3 章では課題および提案手法を示し、4 章では実装について示す。5 章では、本手法の有用性について評価した結果を示す。最後に 6 章でまとめを行う。

2 関連研究

HTML5 アプリケーションをパッケージングし、端末に配信する方法として、Chrome Packaged Apps[1]と、Open Web Apps[2]、PhoneGap[3]がある。

Chrome Packaged Apps は、Chrome ブラウザを実行エンジンとして実行するための HTML5 アプリケーション形体であり、Open Web Apps は、Firefox ブラウザを実行エンジンとするものである。PhoneGap は、WebView とよばれる OS が提供するブラウザ API を実行エンジンとするものである。

これらは、HTML5 アプリケーションを実行する実行エンジンが異なっており、それに起因してパッケージングの作成方法が異なっている。しかし、そのパッケージ内の論理構成は共通する部分が多い。図 1 に示したように、

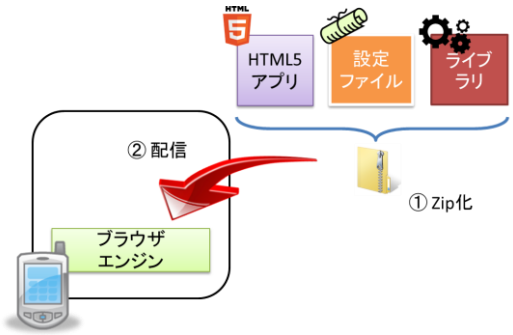

図 1：HTML5 アプリケーションのパッケージ

「HTML5 アプリケーション」に加え、実行エンジン向けの「設定ファイル」、そして、「ライブラリ」が存在する。そしてそれらを zip 化(平文形式)して配信している点が、3つに共通する構成である。

Zip 化しているだけであるため、HTML5 アプリケーション内のデータは保護されていない状態である。この状態を防ぐために、さまざまな手法が用いられる。

よく行われるのは、アプリケーション内にデータ保護のための API を追加し、その API 経由で端末固有のデータ保護機能を呼び出す手法[4]や、実行時に外部サーバからデータを取得する手法 [5]である。この手法の欠点は、アプリケーション開発者が暗号に関するリテラシをもつ必要があり、それが無い場合には、実装ミスから情報漏洩を招く可能性がある点である。

この欠点を克服する方法として、HTML5 アプリケーション全体を暗号化する方法がある。これはパッケージングを行う際に HTML5 アプリケーション全体を暗号化し、そのままの状態で端末に配信する[4][9]。アプリケーションの復号は、アプリケーション体感速度の低下を回避

するために、アプリケーション実行時でなく、配信・インストール時など、アプリケーションの実行が行われる前に、OS によって保護された場所で行われる。

しかし、たとえ OS によって保護された場所であっても、端末のバックアップによってデータが漏洩する事例や、利用者による端末ハッキングによる漏洩事例があり、こういう事例からパッケージ内のデータを保護するためにも、パッケージを暗号化したままデータ読み出しを行えることが望ましい。

3　課題と提案方法

3.1　課題

パッケージの復号は、アプリケーション実行時に行うことがセキュリティ観点で望ましいが、アプリケーションの実行速度が低下するため、ユーザ・エクスペリエンスが損なわれる[9]。このため、実行時における復号を伴うアプリケーション読出速度を解決する必要がある。

3.2　提案手法

そこで、アプリケーション実行時の読出速度を改善するため、読出時に復号処理を行う復号部の端末アーキテクチャへの最適化による高速化と、その復号部にパッケージ内データを最適化することによって、読出速度を向上する手法を提案する(図 2)。

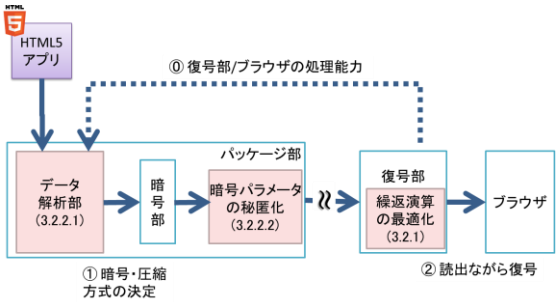

図 2:提案手法

3.2.1　復号部の端末への最適化

復号部で行われる演算には、データ復号処理だけでなく、Zip フォーマット固有の鍵生成・改ざんチェック演算に多くの繰返演算が含まれている。それらを抽出し、端末アーキテクチャを利用して「繰返演算の最適化」を実行することで、高速化を行う。

3.2.2　パッケージ内データの最適化

パッケージ部で行われるパッケージ内データの最適化は 2 つの処理に分かれる。

3.2.2.1　データ解析部

データ解析部は、HTML5 アプリケーションの 1つ1つのデータを解析し、配信しようとしている端末の復号部能力に応じて、読出速度が最大になる暗号・圧縮アルゴリズムを決定する。

3.2.2.2　暗号パラメータの秘匿化

提案手法では、データごとに圧縮・暗号アルゴリズムを最適化するため、1つのパッケージ内に複数の暗号アルゴリズムが混在する。Zip フォーマットでは、暗号パラメータが平文で保存されているため、特定の暗号アルゴリズムの脆弱性とその暗号パラメータから、パッケージ全体の暗号化をほどく事例[6]がある。これに対処するパッケージを行う。

4　実装

ここでは、提案手法の実装について述べる。

4.1　システム構成

図3は、提案方式を検証するために試作した検証システムを表している。

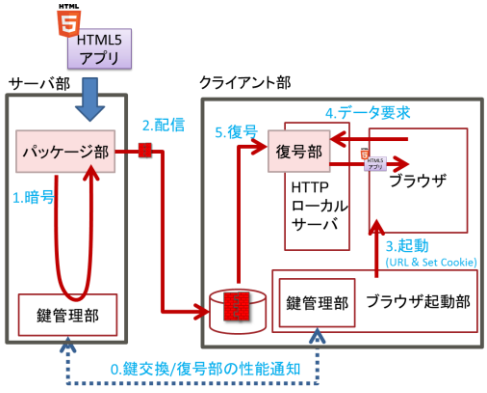

図3:検証システム

サーバ部は、パッケージ部と鍵管理部に分かれ、それぞれservletで実装している。鍵管理部がクライアント側と、鍵および復号部の性能に関する情報交換を行い、パッケージャ部は、その情報にもとづいてHTML5アプリケーションを解析し、解析結果に基づいて暗号・圧縮処理を行う。暗号・圧縮処理には、zip4jを改造したものを用いている。クライアント部は、ブラウザ起動部、ブラウザ、HTTPローカルサーバ部、復号部、鍵管理部に分かれ、それぞれAndroid上のjavaプログラムとして実装している。

動作フロー:鍵管理部がサーバ側と鍵交換を行い、パッケージをほどくための鍵を取得する。サーバからパッケージされたHTML5アプリケーションを取得すると、HTTPローカルサーバ部が、そのパッケージされたデータを読み出すためのWeb APIを生成する。Web APIは、たとえばhttp://127.0.0.1:9800/jktdyz98rd/といった形式のURLになる。Web APIが決定した時点で、ブラウザ起動部がブラウザを初期化(Start URL: http://127.0.0.1:9800/jktdyz98rd/, cookie: ab98hlqage)する。その後、ブラウザがWeb APIからデータを読み出し始めたタイミングで、復号部は鍵管理部から鍵を取得し、パッケージデータを復号しながらブラウザにデータを供給する。

4.2 提案手法の実装

4.2.1 復号部の最適化

図 4 が Zip 仕様[7]で決められている復号処理の内部構造である。

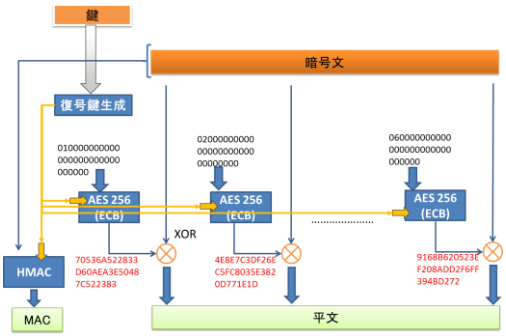

図 4:復号処理の内部構造

パッケージを復号する鍵が1つ存在し、その鍵から「復号鍵生成部」によって各データを復号するための復号鍵と、改ざんチェックを行うための HMAC Key が生成される。復号鍵は、「AES256 ブロック」に供給され、暗号文が平文に復号されていく構成になっている。

この処理のなかで、繰り返し演算が多い部分が、図中で青く塗られた「復号鍵生成部」「HMAC ブロック」「AES256 ブロック」である。

繰り返し演算は、端末の中でもOSに近い部分で処理することによって高速化が見込めるため、この3カ所をAndroidのJNI(Java Native Interface)を通して行い、OS側にあるOpenSSLライブラリを用いて実装することで、復号機能を向上させる。本実装では、復号部はパッケージャ部と同じくzip4jを改造したものを用いた。

4.2.2 データ解析部

パッケージ部のデータ解析部では、HTML5アプリケーションのデータ特性の解析を行う。

ブラウザは、パッケージからデータを読み出す際、そのデータ特性によって、読み出し方が異なっている。たとえば動画データであれば、データをストリームで読み出しながら表示するのに対して、HTML などドキュメント類では、すべてのデータを読み出し終えてから表示する。

この特性を用いて、動画データではランダムアクセス性能が最適になるように配置することで、読み出し性能を向上させるものである。また、データ保護という観点で、パッケージ内データを見てみると、HTML5 アプリケーションには、多くのオープンソースのライブラリが使用されていることがわかる。そこで、オープンソースのデータと、開発者が作成したプログラムやビジネスデータとで暗号アルゴリズムを分け、オープンソースのデータに処理負荷の軽い暗号アルゴリズムを適用することで、読み出し性能の向上を行う。

本実装では、アプリケーション内に含まれているデータのファイル名と拡張子を解析し、それに応じて、読み出しが最適になる暗号圧縮アルゴリズムをファイルとして出力する(図 5)。

```
<?xml version="1.0" encoding="UTF-8"?>
<widget xmlns="http://www.w3.org/ns/widgets">
        <pkg mode="EPKG"  jni="Enable"/>
        <file name="a.html"
                EncMethod=" ENC_METHOD_AES "
                KEYSIZE=""
                compressMethod=" DEFLATE " compressLevel=" ULTRA " />
        <file name="b.pdf"
                EncMethod=" ENC_METHOD_AES "
                KEYSIZE=" AES_STRENGTH_256 "
                compressMethod=" DEFLATE " compressLevel="ULTRA" />
        <file name="c.mp4"
                EncMethod=" ENC_METHOD_AES "
                KEYSIZE=" AES_STRENGTH_256 "
                compressMethod=" STORE " compressLevel="" />
        <file name="jQuery.js"
                EncMethod="ENC_METHOD_STANDARD"
                KEYSIZE=""
                compressMethod=" DEFLATE " compressLevel=" ULTRA " />
        <file name="e.css"
                EncMethod="ENC_METHOD_AES"
                KEYSIZE=" AES_STRENGTH_256 "
                compressMethod=" DEFLATE " compressLevel=" ULTRA " />
</widget>
```

図 5:解析結果の例

このファイルをパッケージャ部が読み出し、ファイル名を比較しながら、それぞれのデータの暗号・圧縮を実行していく。

4.2.3　暗号パラメータの秘匿化

また秘匿化では、図 6 のように Zip フォーマットの暗号化パラメータを1カ所に集めて暗号処理するフォーマット形式で実装を行った。これにより、パッケージをほどく鍵を持つプログラムだけが暗号パラメータを読み取ることができるようになる。

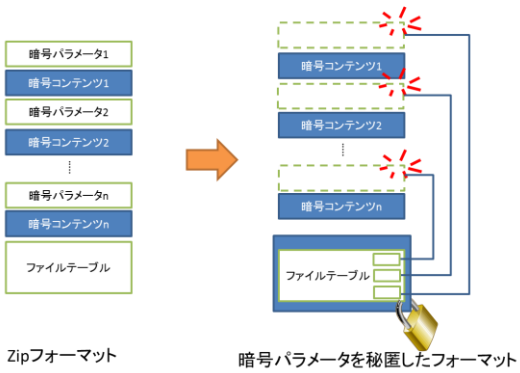

図 6:暗号パラメータの秘匿化フォーマット

5　評価

本章では、前章の検証システムにより、提案手法の有効性を検証する。

測定に用いた端末は、Nexus 5(Android 4.4.4 CPU: Quad-core 2.3 GHz Krait 400)である。

5.1　復号部最適化の評価

評価のため、40MB のパッケージデータを暗号アルゴリズム別に用意し、ブラウザが復号データ読出に要した時間を測定した。図 7 は、復号機能を端末に最適化していないときの、読出速度の比較グラフである。Zip(平文)と比べて約14～18倍の時間、読出に時間を要しているのがわかる。

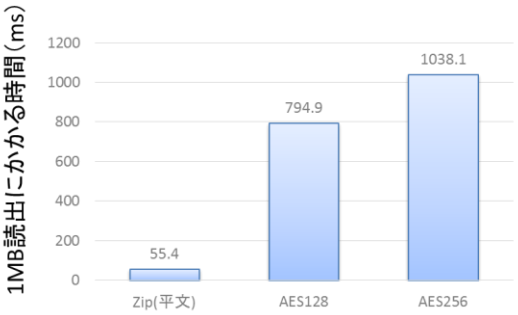

図 7:最適化前のデータ読出速度比較

一方、復号部の最適化を行い、読出を行った結果が図 8 である。zip(平文)の 1.3 倍～1.4 倍の読出速度を達成できている。また図7と比べて読出時間が約 1/10 に短縮できていることがわかる。

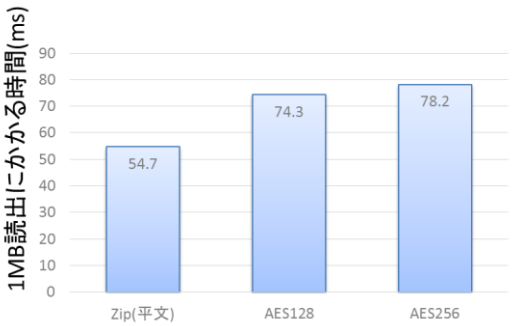

図 8:最適化後のデータ読出速度比較

5.2　データ解析部の評価

パッケージ最適化の効果を見るために、一般 Web サイトにある HTML5 アプリケーションを取得(**9leap.net より取得した 3MB のゲームアプリ。全ファイル数は 30 ファイル**)し、最適化を行ったものと、最適化を行わず同一暗号・圧縮アルゴリズム(AES256, Deflate)でパッケージ化したものを用意し、初期ページ表示までに要した時間(loadEventEnd – navigationStart で計算)[8]を測定した(図 9)。

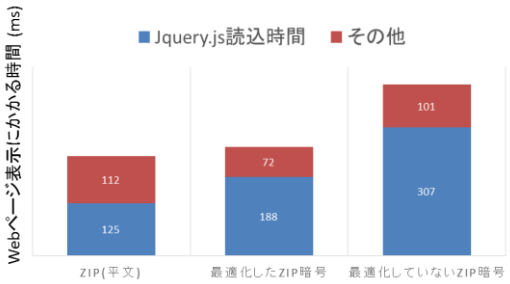

図 9:最適前後によるページ読出速度比較

測定対象のアプリケーションには、オープンソースとして JQuery.js が入っており、最適化処理によって、このプログラムの暗号レベルが AES256→PKZIP に変更されている。図 9 より、初期ページ読出時間が最適化前と比較して約 40%改善され、zip(平文)と比較しても約 10%の負荷で読出が可能となっていることがわかる。

また、最適化による動画読み出しの効果をみるために、mp4 動画(映像の長さは 1 分)の映像ビットレートを変化させたパッケージを複数用意し、それぞれを最適化前と後とで体感速度を評価した(図 10)。最適化することによって、平文と比較して数百 ms の遅延で再生できているのに対して、“最適化をしていない”場合には、再生ボタンを押してから映像が再生されるまでの遅延が、ビットレートとともに高くなり、4Mbps を超えた時点で読出速度が不足して、映像がスムーズに再生できなくなっていた。

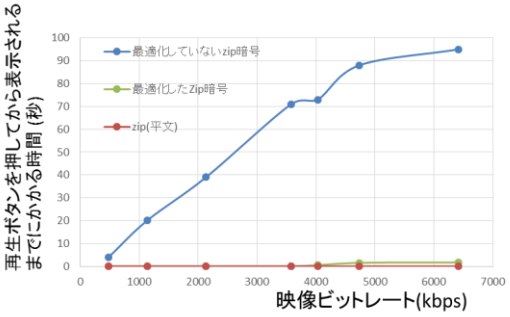

図 10:最適化前後による動画読出性能比較

このことから、データ特性に合わせて暗号パッケージ内データを最適化することにより、平文と同じ体感速度で動画再生ができることがわかる。

6 まとめ

本稿では、パッケージを暗号化したまま、アプリケーション実行の体感速度を向上させるため、復号部の高速化、データ特性による暗号・圧縮レベルの変更を行う手法を提案した。また、検証システムを構築し、提案手法によって、暗号化したままのパッケージで、Zip(平文)パッケージの約 1.1 倍の読出速度を実現できることを示した。

参考文献

[1] Chrome Packaged Apps 仕様: https://developer.chrome.com/apps/first_app

[2] Open Web Apps 仕様: https://developer.mozilla.org/ja/docs/Web/Apps/Packaged_apps

[3] PhoneGap 仕様:http://phonegap.com/

[4] IBM Worklight 資料: http://www-06.ibm.com/software/jp/websphere/events/livestream/Worklight_System_Design_05.pdf

[5] Encrypted Media Extensions 仕様: https://dvcs.w3.org/hg/html-media/raw-file/tip/encrypted-media/encrypted-media.html

[6] Eli Biham, Paul C. Kocher:「A Known Plaintext Attack on the PKZIP Stream Cipher」

[7] Zip 仕様 http://www.pkware.com/documents/ casestudies/ APPNOTE.TXT

[8] Navigation Timing 仕様 http://www.w3.org/TR/navigation-timing/

[9] Kazuaki Nimura, Hidenobu Ito, Yousuke Nakamura, Kouich Yasaki: A Secure Use of Mobile Application with Cloud Service, SmartApp (International Workshop on Smart Mobile Application), June, 2012, http://www. mobile. ifi. uni-muenchen.de/aktuelles/archiv/smartapps2012/smartapps12_secure.pdf