誤り訂正符号に基づく偽装QRコードの検出手法の実装

川口 宗也[1,a)]　小林 海[1]　木村 隼人[1]　鈴木 達也[2]　岡部 大地[1]　石橋 拓哉[1]　山本 宙[1]　大東 俊博[1]
乾 真季[3]　宮本 亮[3]　古川 和快[3]　伊豆 哲也[3]

概要：2018 年 6 月の ICSS 研究会にて大熊らは確率的に悪性サイトに誘導可能な QR コード（偽装 QR コード）の構成法を示した．この方法は QR コードに使われている誤り訂正符号の性質に注目しており，復号誤りが生じやすいように改変した QR コードに対してカメラによる白黒検出の誤りを誘発する汚れを付加することで確率的な挙動の変化を実現している．2020 年 3 月の CSEC 研究会にて大東らは偽装 QR コードの攻撃手法の性質に注目し，読み取った QR コードの誤り訂正符号の復号時の誤り訂正数や誤り位置，汚れなどを想定した誤り方から，正規の QR コードか偽装 QR コードかを識別する偽装 QR コード検出手法を提案している．しかしながら，この手法は理論的に考察されたのみであり，実際にソフトウェア上での有効性は確認されていない．そこで，本稿では 3 種類の偽装 QR コード検出手法を Android アプリケーションとして実装し，偽装 QR コードを検出できることを実証する．さらに複数の検出手法を併用した判定方法を導入することで，単独の検出手法では誤検出をしてしまうような「汚れが付加された正規の QR コード」の多くを正しく判定できることも示す．

キーワード：QR コード，偽装 QR コード，誤り訂正符号

Implementation of Detection Methods for Fake QR Codes Based on Error-Correcting Codes

Shuya Kawaguchi[1,a)]　Kai Kobayashi[1]　Hayato Kimura[1]　Tatsuya Suzuki[2]　Daichi Okabe[1]
Takuya Ishibashi[1]　Hiroshi Yamamoto[1]　Toshihiro Ohigashi[1]　Maki Inui[3]　Ryo Miyamoto[3]
Kazuyoshi Furukawa[3]　Tetsuya Izu[3]

Abstract: In 2018, Okuma et al. have proposed fake QR codes, which can guide users to a malicious web page probabilistically. The fake QR codes are based on a miscorrection of Error-Correcting Codes and the misidentification of the camera device. In March 2020, Ohigashi et al. proposed a detection methods for fake QR codes, which focuses on the number of symbols recovered by error-correcting codes, the positions of error symbols, and the error patterns of the QR code image. In this paper, we implement the detection methods for fake QR codes as an Android software, and evaluate the validity of the detection methods using our software.

Keywords: QR code, Fake QR Codes, Error-Correcting Codes

1. はじめに

二次元バーコードの一種である QR コードは携帯電話やスマートフォンの普及に伴い，ウェブサイトへのアクセスや決済等，情報伝達の手段として幅広く利用されるようになった．QR コードはデコーダを用いることによって多くの情報を素早く読み取ることができるが，白と黒のモジュールが並べられた画像データでしかないため，人間が直接データを読み取る事が困難である．これを利用して悪意のある攻撃者は悪性サイトの URL を埋め込んだ QR コードを掲示することにより，その QR コードを読み込ん

[1] 東海大学, Tokai University
[2] 筑波大学, University of Tsukuba
[3] 富士通研究所, FUJITSU LABORATORIES LTD.
[a)] 7bjt2204@mail.u-tokai.ac.jp

だ利用者を悪性サイトへ誘導してマルウェアの配布を行うといった攻撃が可能となる．一般的な QR コードの作成方法に従って悪性サイトの URL が埋め込まれた QR コードであれば，常に悪性サイトに誘導されるため発見が容易であり，早い段階で被害を抑えることができる．

2018 年 6 月の ICSS 研究会および一連の発表において大熊らは確率的に挙動が変化する QR コードの存在と具体的な構成法を示した [1], [2], [3], [4], [5]．大熊らの手法によって作られた QR コード（偽装 QR コードと呼ばれる）では，悪性サイトへの誘導確率を 0.6%のような極端に小さい値にすることも可能である．このような QR コードはほとんどの場合で悪性サイトには誘導されないため，偽装に気づきにくく，被害が拡大してしまうことが予想される．

これに対し 2020 年 3 月開催の CSEC 研究会にて著者らは，偽装 QR コードの構成法の原理に注目した方法により，掲示されている QR コードが偽装 QR コードか否かを判定する検出手法を提案している [6]．提案した手法は以下の 3 種類に分類される．

誤り数に注目した方法： 悪性サイトへの誘導に失敗した場合でも誤り訂正符号によって訂正される誤りの個数が大きくなることに注目した方法

誤り位置に注目した方法： 確率的に誘導するために誤り訂正符号によって訂正される誤りの位置が誤り訂正ブロックに集中することに注目した方法

誤り方に注目した方法： 自然に誤りが生じる場合と偽装 QR コードで人為的に誤らせる場合の誤り方の違いに注目した方法

しかしながらこの論文では検出手法の提案を行っただけであり，実装による動作確認は行っていない．そこで，本研究では文献 [6] で提案された偽装 QR コードの検出手法を Android アプリケーションとして実装し，提案検出手法が実際に偽装 QR コードを正しく検出できることを示す．具体的には，Android 用の QR コードリーダの OSS ライブラリである zxing*1 をベースにして 3 種類の偽装 QR コードの検出手法をそれぞれ 1 つずつ実装している．さらに，偽装 QR コードの検出手法を複数併用する判定方法を導入することによって，単一の検出手法だけでは誤検出になってしまうような汚れが付加された正規の QR コードの多くを偽装 QR コードでは無いと判定できたことも報告する．

2. 準備

2.1 QR コード

QR コードは 1 ビットの情報を示すシンボルと呼ばれる最小単位のセルを 2 次元に配置し，情報の埋め込み・読み込みを行う．配置できるモジュールの数は型番（version）によって指定でき，例えば version 2 では 25x25 モジュールを配置できる．また，QR コードは汚れ等があったときにも正確に読み取れるようにするために誤り訂正符号を

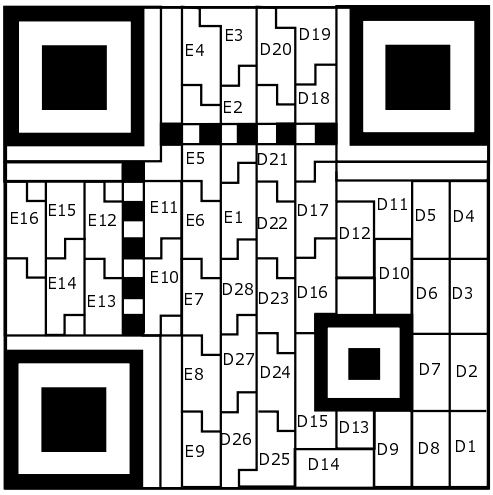

図 1 QR コード（2-M 型）のコード配置

使っており，その誤り訂正レベル（EC レベル）を L，M，Q などで指定する．QR コードは数字データ，英数字データ，8 ビットバイトデータ，漢字データなどが扱えるが，どの種類のデータであるかはモード指示子と呼ばれる 4 ビットの識別子で指定する．例として，8 ビットバイトデータ（モード識別子は 2 進数で $(0100)_2$）で 2-M 型（version 2, EC レベル M）の QR コードのデータフォーマットを以下に示す．

- モード指示子（4 ビット）: $(0100)_2$
- 文字数指示子（8 ビット）: データ文字列の長さを 2 進数で指定
- データ文字列（最大 26 バイト）: 文字列指示子で指定したサイズのデータ本体
- 終端パターン（4 ビット）: 0 でパティング
- 埋め草コード: データ文字列のバイト数が最大値である 26 バイトに満たない場合に特定のパターン（236, 17 の繰り返し）でパディング
- 誤り訂正ブロック（16 バイト）: 上記の 5 つのデータを用いて Reed-Solomon(RS) 符号で計算した冗長ビット

最初の 5 つのデータ（28 バイト分）を D1〜D28，誤り訂正ブロックを E1〜E16 としたときの 2-M 型の QR コードのコード配置を図 1 に示す．各バイトに 8 つのモジュール（ビット）が確保されていることがわかる．また，QR コードを撮影時に向きを合わせたり，モジュールの走査線を得るために，QR コードでは上記のデータ以外に「位置検出パターン」や「位置合わせパターン」という特定のパターンを配置している．

QR コードでは RS 符号を用いて誤りが生じた際に正しいデータを復元する．RS 符号は符号語（冗長ビットを付加された系列）をビットの集まり（シンボル）単位で表わし，シンボル単位で誤り訂正を行うため連続して起こるビット単位の誤り（バースト誤り）に強いという特徴を持つ．QR コードでは 1 バイトを 1 シンボルとして符号化を

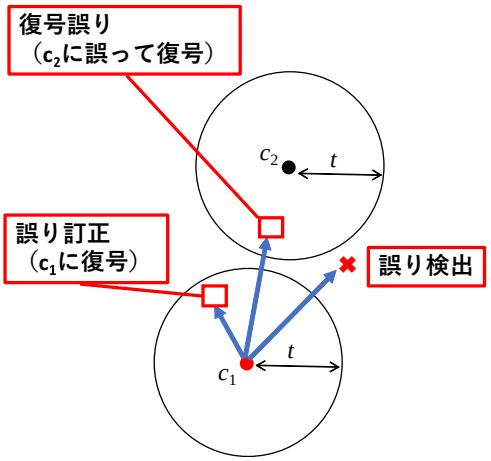

図 2　誤り訂正，誤り検出，復号誤り

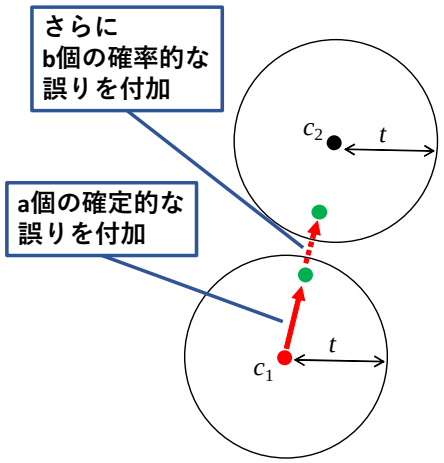

図 3　偽装 QR コードの原理

行っており，図 1 のように連続したビットを隣接したモジュールに配置することでマジック等で連続した複数のモジュールが汚れた場合でも少数のシンボルの誤り訂正で対応できる．符号長 n と情報点数 k とした (n, k) RS 符号において，ユークリッド復号法を用いて訂正可能な誤りの数は設計距離 $d = n - k + 1$ により保証されており，訂正可能な誤りシンボルの数は $t = \lfloor \frac{d-1}{2} \rfloor = \lfloor \frac{n-k}{2} \rfloor$ で与えられる．QR コードでは誤り訂正ブロック以外のデータのサイズが k バイト，誤り訂正ブロックを含んだ全体のサイズが n バイトという対応になり，2-M 型では $n = 44$，$k = 28$，$t = 8$ となる．誤り訂正能力 t の符号を用いた場合，図 2 のように t シンボルまでの誤りまで訂正可能であり，t シンボルを超えた誤りが生じたときには誤り検出または復号誤りが生じる．t シンボルを超えて誤ったときに別の符号語の t シンボル以下の誤った受信後になっていた場合にはその受信語に誤って復号されてしまい（復号誤り），どの符号語にも復号できない受信語になっている場合には誤り検出は行えるが復号はできない．

2.2　偽装 QR コード

2.2.1　原理

大熊らは復号誤りを誘発する QR コード（偽装 QR コード）によって確率的に悪性サイトへアクセスさせる方法を示している [1], [2], [3].

正規のサイトの URL を含んだ QR コード用の符号語を c_1，ある悪性サイトの URL を含んだ QR コード用の符号語を c_2，二つの符号語の距離（異なるシンボルの個数）を $d(c_1, c_2)$ とする．このとき，まず初めに c_1 よりも c_2 との距離が $a(\leq t)$ 小さくなるように a 個の誤りを付加した系列 c_1' を作成する．系列 c_1' から作成された QR コードは誤りに対する耐性が低くなるが，t 個以下の誤りしか含んでいないため撮影環境が良好であれば正規のサイトの URL が復号される．さらに系列 c_1' よりも c_2 との距離が b 小さくなるように b 個の誤りを更に付加して系列 c_1'' を作成する．ここで，b は $d(c_1, c_2) \geq a + b \geq d(c_1, c_2) - t$ を満たすとする．このとき，系列 c_1'' は誤り訂正によって c_2 に復号される（復号誤り）ことから，系列 c_1'' から作成された QR コードでは撮影環境が良好であれば悪性のサイトの URL が復号される．偽装 QR コードでは a 個の誤りは対応するモジュールの白黒を反転させることで「確定的な誤り」を付加し，残りの b 個の誤りは「確率的な誤り」を付加するというテクニックで作成される．b 個の誤りは確率的に生じることから，誤らなかった場合には正規サイトにアクセスし，誤りが生じた場合には悪性サイトにアクセスさせるといった確率的な悪性サイトへの誘導が実現される．図 3 に偽装 QR コードと誤りの与え方について示す．

偽装 QR コードを実現するために用いる確率的な誤りを発生させる方法は 2 種類提案されている．文献 [1], [2], [3] ではモジュールの中心部の輝度値を変更すること，文献 [5] ではモジュールの位置をずらして QR コードのデコーダプログラムで白黒判定を行う際の走査線の位置をずらすことによってカメラでの QR コードの読み取り後の白黒判定処理が不安定になって誤りが確率的に生じるようになる．特に文献 [1] の実験では，確率的な誤りを生じさせるモジュール（白色）の中心に打つ黒色の点の輝度値を 0 から 255 まで段階的に変更した結果，輝度値 0 では誤る確率が 65.8% であったのが輝度値 160 では 0.6% という低確率での誤りが生じたことが報告された．

2.2.2　偽装 QR コードの作成手順

ここで正規サイトとして `http://www.u-tokai.ac.jp/`，悪性サイトとして `http://www.u-yokai.ac.jp/` を例にして偽装 QR コードの作成手順を説明する．このとき，正規サイトの符号語は

$$\begin{aligned} c_1 = (&65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114,\\ &231, 82, 215, 70, 246, 182, 22, 146, 230, 22, 50, 230,\\ &167, 2, 240, 236, 36, 111, 113, 168, 84, 82, 21, 223,\\ &143, 148, 4, 29, 86, 247, 145, 151) \end{aligned}$$

悪性サイトの符号語は

$$
\begin{aligned}
c_2 \;=\; (&65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114,\\
&231, 82, 215, \mathbf{150}, 246, 182, 22, 146, 230, 22, 50, 230,\\
&167, 2, 240, 236, \mathbf{207}, \mathbf{252}, \mathbf{132}, \mathbf{239}, \mathbf{96}, \mathbf{205}, \mathbf{117}, \mathbf{61},\\
&\mathbf{38}, \mathbf{171}, \mathbf{42}, \mathbf{232}, \mathbf{175}, \mathbf{206}, \mathbf{111}, \mathbf{215})
\end{aligned}
$$

となる．上記の c_2 の符号語の表記で太字にしているのは，c_1 と c_2 でシンボルが異なっているバイトである．URL で 1 文字変更したことで生じた 1 バイト分の差分とそれによって誤り訂正ブロック 16 バイト全てに差が生じたことから，c_1 と c_2 の距離は $d(c_1, c_2) = 17$ となる．なお，これは設計距離 $d = n - k + 1$ と一致しており，符号語間の距離は最小になり，復号誤りを生じさせやすい符号語のペアと言える．

偽装 QR コードを作成するために a 個の確定的な誤りと b 個の確率的な誤りを与える位置を決定する．文献 [1] に示された例に合わせて $a = 8$，$b = 1$ とする．RS 符号で復号可能な限界である 8 シンボルの誤りを与えて，追加で 1 シンボル誤る場合に符号語 c_2 への復号誤りを生じさせる．確率的な誤りは変化させるビット数が多いほどモジュールに変化を加える数が増えることから，c_1 と c_2 の異なる箇所の中でもハミング距離が小さいものを選択したい．その結果，確定的な誤りは E1〜E8，確率的な誤りを生じさせる箇所はハミング距離が 1 であった E16 にした．c_1 の E16 は $151 = (10010111)_2$，c_2 の E16 は $215 = (11010111)_2$ であり 2 ビット目だけ異なっており，そこに対応するモジュールを確率的に変化させる．確率的に変化するビットを ? の表記として，E16 にセットする値を $X = (1?010111)_2$ とした場合に，偽装 QR コードのための系列 c_1'' は以下のように与えられる．

$$
\begin{aligned}
c_1'' \;=\; (&65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114,\\
&231, 82, 215, 70, 246, 182, 22, 146, 230, 22, 50, 230,\\
&167, 2, 240, 236, \underline{207, 252, 132, 239, 96, 205, 117, 61},\\
&143, 148, 4, 29, 86, 247, 145, \underline{X})
\end{aligned}
$$

なお，下線を引いたところが c_1 からの変更した箇所である．図 4 に c_1'' を用いて作成した QR コードを例として示す．確率的な誤りはモジュールの中心部の輝度値を変更する方法を採用し，白色 (0) を黒色 (1) に誤認識させるようにモジュールの中心の 1 ピクセルを輝度 0 に変更している．

上記の例では誤り訂正の範囲外に出るときに「誤り検出」でエラーが生じることが無いように必ず「復号誤り」が生じるように確率的な誤りを 1 ビットだけに適用していた．しかしながら，QR コードリーダでは誤り検出を起こした際にエラー出力をするのではなくリトライをする実装になっていることから，確率的な誤りを複数のシンボルやビットに付加しても偽装 QR コードは正規のサイトか悪性サイトのどちらかにアクセスするように動作する．ただし，確率的な誤りを与えるビット数（改変するモジュール数）を増やしたときには誤り検出時のリトライが増え，読み込みが成功するまでの時間が増加するというデメリットもある．

図 4 偽装 QR コードの例

3. 偽装 QR コードの検出手法

文献 [6] で著者らが提案した 3 種類の偽装 QR コードの検出手法について解説する．この検出手法は偽装 QR コードの構成法に注目し，読み取った QR コードの誤り訂正符号の復号時の誤り数や誤り位置，汚れなどを想定した誤り方から正規の QR コードか偽装 QR コードかを識別する．

3.1 誤り数に注目した手法

本節では QR コードを撮影後に RS 符号で復号する際に訂正された誤りの個数に注目した検出手法を説明する．QR コードで用いられている RS 符号の復号は

Step 1 受信系列からシンドロームを求める．
Step 2 シンドロームを用いて，誤り位置多項式を導出．
Step 3 誤り位置多項式を用いて，誤り位置を計算する．
Step 4 誤り評価多項式を用いて，誤っている数値の正しい数値を計算する．
Step 5 求められた誤り位置と正しい数値を用いて訂正を行い，復号結果を得る．

の順に実行され，Step 3 で得られる誤り位置から誤っている個数を得ることができる．したがって，QR コードのデコード時に誤りの個数を用いた偽装 QR コードの検出手法を実行することが可能となる．

偽装 QR コードでは確率的に誤るビット数が多くなると「誤り検出」の判定によるリトライが繰り返されて読み込みの時間が増加することから，確定的な誤りの個数 a を大きくし，確率的な誤りの個数 b を小さくすることになる．したがって，偽装 QR コードを読み込んだ場合，悪性サイトへ誘導されない場合でも誤り数は常に a 以上の大きなものとなる．これは正規の QR コードでは起こりにくい現象であり，正規の QR コードなのか偽装 QR コードなのかを判別する際に用いることができる．少なくとも QR コード

が破損や汚れが無く，明るい場所などで安定して誤り数が多いようなことがあれば，偽装 QR コードである可能性が高いと判断できる．

仮に確定的な誤りの個数 a を小さくし，確率的な誤りの個数 b を大きくできる偽装 QR コードが作られたとしても，その状態で現実的な確率で悪性サイトに誘導するためには復号時の誤りの個数が頻繁に変化するという挙動が生じ，それに注目することにより検出が可能と考えられる．この詳細は文献 [6] で議論している．

3.2 誤り位置に注目した手法

本節では QR コードを撮影後に RS 符号の復号する際に訂正された誤りの位置が誤り訂正ブロックに集中することに注目した検出手法を説明する．**3.1** 節で示した RS 符号の復号処理の Step 3 によると，誤り位置多項式を用いて誤り位置を取得できる．したがって，QR コードのデコード時に誤り位置を用いた偽装 QR コードの検出手法を実行することが可能となる．

偽装 QR コードでは復号誤りを起こさせるための確率的な誤りの個数を少なくするために，正規サイトの符号語 c_1 と誘導する悪性サイトの符号語 c_2 の距離 $d(c_1, c_2)$ を小さくする必要がある．文献 [1] では正規サイトの URL を 1 バイト変更したものを悪性サイトの URL として偽装 QR コードを作成する方法が示されている．この場合，URL 部分で 1〜2 シンボルの差分が生じ，それ以外の差分は誤り訂正ブロックに生じる．これらの差分の個数である $d(c_1, c_2)$ は RS 符号の設計距離 $d = n - k + 1$ に近いものとなる．このように c_1 と c_2 の差分は誤り訂正ブロックに集中している．

c_1 から c_1'' を作成するために付加する確定的な誤りや確率的な誤りは，c_2 に近づくように誤らせるために符号語同士に差分がある箇所を移植することになる．そのため，c_1'' と c_1 の差分は誤り訂正ブロックに集中することになる．このとき，確率的な誤りが生じなかった場合には c_1 へ訂正されるため移植した c_2 のシンボル（**2.2.2** 節で言えば c_1'' の下線部分）が誤りとして訂正され，確率的な誤りが生じた場合には c_1 へ訂正されるため移植した c_2 のシンボル以外の c_1 と c_2 の差分（**2.2.2** 節で言えば c_1'' の下線部分以外の c_2 の太字部分）が誤りとして訂正される．これらは双方ともに大部分が誤り訂正ブロックである．したがって，誤り位置が誤り訂正ブロックである割合を利用することにより，偽装 QR コードを検出できる可能性がある．

3.3 誤り方に注目した手法

用紙に印刷された正規の QR コードに誤りが生じるとき，マジックで文字を書くなどして特定の範囲に色がついてしまう，用紙の一部が破れて白色になってしまう，撮影時にカメラの前に指がかかってしまうなど連続した誤りが生じるケースが想定される．一方，偽装 QR コードの確率的な誤りではモジュールの中心だけ色が変わるような特徴的な汚れの付き方で誤りを生じさせている．このように自然な QR コードの使い方での誤りの生じ方と偽装 QR コードでの誤りの生じ方で差異があり（図 5），その差異を偽装 QR コードの検出に役立てることができる．具体的には，モジュールをまたいだ誤りが生じた際に，撮影した QR コードの画像を用いた画像処理により誤りを生じさせている汚れがモジュールをまたいで連続しているかを確認をする方法が示されている．

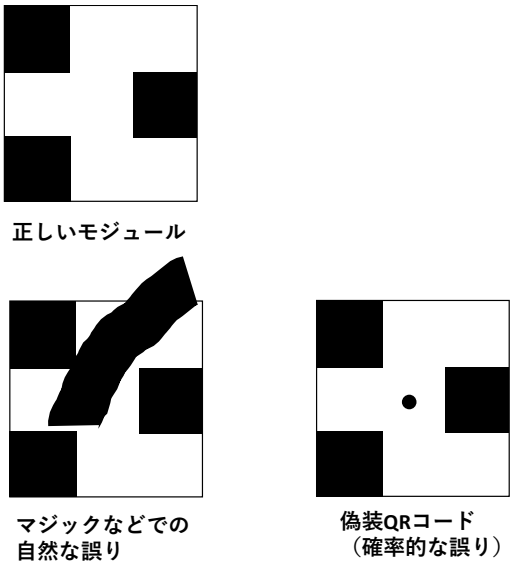

図 5 モジュールの誤り方の違い

文献 [6] では別の方法として，破れやマジックなどでの汚れではある範囲のモジュールが全て 1（黒）になるか 0（白）になるような誤り方をすることに基づく検出手法も示されている．偽装 QR コードでは，悪性サイトの誤り訂正ブロックを移植することによる誤りの付加の際に，モジュール単位でみると白黒が交互に誤るなどの通常の汚れ方ではあり得ない誤りが検出されて訂正される．例えば，**2.2.2** 節の例では，c_1 の E1 は $36 = (00100100)_2$，c_1'' の E1 は $207 = (1101111)_2$ であり黒 (1) から白 (0) への誤りと白 (1) から黒 (0) への誤りが近いモジュールで複数見られる．他の誤りでも同様となっていることから，バースト的に全体が白や黒に変化するような特徴的な誤りと区別でき，それを用いて偽装 QR コードの検出に利用できる．

4. 偽装 QR コードの検出方法の実装

文献 [6] で提案された偽装 QR コードの検出手法を 1 種類ずつ Android アプリケーションとして実装する．実装には Android 用のライブラリである zxing を用いる．さらに，これらの検出手法を用いて偽装 QR コードを判定する判定手法についても提案する．

4.1 誤り数に注目した検出手法

誤りの個数に注目した手法は誤りの個数を閾値として用いる方法と撮影ごとの誤りの個数の変化量から検出する方法が示されている．本節では前者の誤りの個数を取得して一定以上の数値の場合に誤りと判定する方法を実装した．

誤り数から偽装 QR コードを検出する方法では，誤り訂正可能な誤りの最大数 t に対する誤り数の割合の大きさから偽装 QR コードか否かを判断する．**2.1** 節で説明したとおり，最大誤り数 t は誤り訂正ブロックを含んだ QR コードの全体のバイト数 n と誤り訂正ブロック以外のデータバイト数 k から $t = \lfloor \frac{n-k}{2} \rfloor$ のように求めることができる．n, k は QR コードの version と EC レベルによって一意に決定されるため，QR コード読み取り時のフォーマット情報から version と EC レベルを取得して，最大誤り数 t を求めている．読み込み時の誤り数は RS 符号の復号過程で得られる誤り位置多項式から求めている．以下に検出手法の具体的な手順を示す．

Step 1 0〜1 の範囲内でしきい値を設定する．

Step 2 対象の QR コードから version 情報及び EC レベルを取得する．

Step 3 取得した version 情報と EC レベルから n, k を取得し，誤り訂正可能な最大誤り数 t を計算する．

Step 4 RS 符号の復号過程から誤り数を取得する．

Step 5 誤り数を t で割った値を**誤り数に注目した検出手法のスコア**とし，そのスコアがしきい値を超えていれば偽装 QR コードと判断する．

4.2 誤り位置に注目した検出手法

誤り位置から偽装 QR コードを検出する方法では，QR コード全体での誤りに占める誤り訂正ブロックの割合の大きさから偽装 QR コードか否かを判断する．**4.1** 節の Step 4 により誤り数を取得し，その際に得られる誤り位置多項式から誤り訂正ブロックに該当する誤り数も求めることができる．以下に検出手法の具体的な手順を示す．

Step 1 0〜1 の範囲内でしきい値を設定する．

Step 2 対象の QR コードから version 情報と EC レベルを取得する．

Step 3 取得した version 情報と EC レベルから誤り訂正ブロックを含んだ全体のバイト数 n と誤り訂正ブロック以外のバイト数 k を取得し，誤り訂正ブロックの範囲（何バイト目から何バイト目かという情報）を把握する．

Step 4 RS 符号の復号過程から誤り数と誤ったバイトの位置を取得し，Step 3 で得られた情報から誤り訂正ブロック内の誤り数を求める．

Step 5 誤り訂正ブロック内の誤り数を全体の誤り数で割った値を**誤り位置に注目した検出手法のスコア**とし，スコアがしきい値を超えていれば偽装 QR コードと判断する．

この検出手法と **4.1** 節の検出手法を実現するためには QR コード読み込み時の RS 符号の復号過程から誤り位置多項式を取得し，誤り数と誤ったバイトの位置を得る必要がある．この検出手順を zxing に組み込む方法について概説する．zxing において QR コードの読み込み後に RS 符号の復号過程の誤り数等の情報は **ReedSolomonDecoder** クラスの **decode** 関数の中で取得が可能である．我々の実装において修正した **decode** 関数を以下に示す．

```
1  public void decode(int[] received, int twoS) throws
       ReedSolomonException {
2          MyGenericGFPoly poly = new MyGenericGFPoly(field,
               received);
3          int[] syndromeCoefficients = new int[twoS];
4          boolean noError = true;
5          for (int i = 0; i < twoS; i++) {
6              int eval = poly.evaluateAt(field.exp(i + field
                   .getGeneratorBase()));
7              syndromeCoefficients[syndromeCoefficients.
                   length - 1 - i] = eval;
8              if (eval != 0) {
9                  noError = false;
10             }
11         }
12         byte[] rawByte = new byte[received.length];
13         for (int i = 0; i < received.length; i++){
14             rawByte[i] = (byte) received[i];
15         }
16         if (noError) {
17             MyDataClass.setErrorCount(0);
18             return;
19         }
20         MyGenericGFPoly syndrome = new MyGenericGFPoly(
               field, syndromeCoefficients);
21         MyGenericGFPoly[] sigmaOmega =
22                 runEuclideanAlgorithm(field.buildMonomial(
                       twoS, 1), syndrome, twoS);
23         MyGenericGFPoly sigma = sigmaOmega[0];
24         MyGenericGFPoly omega = sigmaOmega[1];
25         int[] errorLocations = finderrorLocations(sigma);
26         int[] errorMagnitudes = findErrorMagnitudes(omega,
               errorLocations);
27         int[] errorPositions = new int[errorLocations.
               length];
28
29         int errorCount = sigma.getDegree();
30         MyDataClass.setErrorCount(errorCount);
31
32         for (int i = 0; i < errorLocations.length; i++) {
33             int position = received.length - 1 - field.log
                   (errorLocations[i]);
34             if (position < 0) {
35                 throw new ReedSolomonException("Bad_error_
                       location");
36             }
37             errorPositions[i] = position;
38             received[position] = MyGenericGF.addOrSubtract
                   (received[position], errorMagnitudes[i]);
39         }
40         for (int i = 0; i < received.length; i++){
41             rawByte[i] = (byte) received[i];
42         }
43         Arrays.sort(errorPositions);
44         MyDataClass.setErrorPositions(errorPositions);
45     }
```

紙面の都合上詳細は省略するが，誤り数は 29 行目の sigma.getDegree(); で取得し，誤り位置は 33 行目で取得して 37 行目で配列に格納している．

4.3 誤り方に注目した検出手法

誤り方から偽装 QR コードを検出する方法として，誤ったシンボルの内，白→黒になる誤りと黒→白になる誤りが混在しているシンボルがどの程度存在するかによって偽装 QR コードか否かを判断する方法の実装を行った．以下に検出手法の具体的な手順を示す．

Step 1 0〜1 の範囲内でしきい値を設定する．

Step 2 対象の QR コード（以下 QR コード A）を復号する．この際に RS 符号の復号過程から誤り数と誤ったバイトの位置を取得しておく．

Step 3 復号した情報及び QR コード A のフォーマット情報と読み取ったデータを用いて，QR コード A が

誤っていなかった場合の QR コード（以下 QR コード B）を生成する．

Step 4 QR コード A, B からマスク[*1]を取り除く処理を行わずにデータを読み取る．

Step 5 Step 2 で取得した誤ったバイトの位置を選択し，その位置のデータ同士で排他的論理和をとり変数 x に代入する．

Step 6 変数 x と QR コード A の誤ったバイトの位置のデータで論理積をとったものを変数 y に代入する．

Step 7 変数 x と QR コード B の誤ったバイトの位置のデータで論理積をとったものを変数 z に代入する．

Step 8 y と z が共に 0 より大きい場合，白と黒の汚れが混在していると判断する．

Step 9 Step 5〜Step 8 までの操作を誤ったバイトの位置の全てについて実行し，その後 Step 10 へ．

Step 10 Step 8 で白と黒の汚れが混在していると判断されたシンボル数を誤り数で割った値を**誤り方に注目した検出手法のスコア**とし，スコアがしきい値を超えていれば偽装 QR コードと判断する．

4.4 複数の検出手法を併用した偽装 QR コードの判定方法

偽装 QR コードは実装した 3 種類の検出手法の全てで判定される特徴を持つため，どの方法を用いても検出できる．しかしながら，検出手法を単独で用いる場合，偽装 QR コードではない正規の QR コードでも汚れの付き方によっては偽装 QR コードと誤検出してしまうことが予想される．例えば，誤り数に注目した検出手法を適用した場合，正規の QR コードにマジック等で広範囲の汚れをつけてしまっただけで誤り数は増大し，偽装 QR コードと検出してしまう可能性がある．

誤検出については，偽装 QR コードの恐れがあることと，広範囲に汚れがついている場合に誤検出の可能性があることをアプリケーション利用者に提示することで対応が可能であると考えているが，そのメッセージが頻繁に出るのは避けたい．そこで誤検出の頻度を抑えるために，実装したアプリケーションでは複数の検出手法のスコアを利用して偽装 QR コードを判定する方法を導入する．具体的には，誤り数，誤り位置，誤り方注目した検出手法の全てのスコアがしきい値を超えた場合に偽装 QR コードであると判定する方法である．例えば，上記のマジックによる一定の汚れでは誤り方の検出手法が反応しないため，誤検出を防ぐことが可能となる．

[*1] データを配置した状態で一方の色のモジュールが極端に多かったり，位置検出パターンに類似した模様がある場合に読み取りに支障をきたさないために適用するもの．

表 1 動作環境

端末	HUAWEI nova lite 2
OS	EMUI9.1.0[*2]
ビルド番号	9.1.0.205（C635E6R1P5）
CPU	Hisilicon Kirin 659
メモリ	3.0GB

5. 評価実験

開発したアプリケーションが偽装 QR コードを実際に検出できるのか，汚れが付加された正規の QR コードの誤検出を防ぐことができるのかを評価するための実験を行った．

5.1 実験条件

評価実験のために偽装 QR コードを含むいくつかのパターンの QR コードを図 6〜図 10 のように作成した．図 6 は偽装 QR コードであり，開発したアプリケーションが偽装 QR コードを検出できるかの評価に用いる．誤検出に関する評価のために，汚れがついていない正規の QR コード（図 7），正規の QR コードに単色の小さい汚れを付加したもの（図 8），正規の QR コードに黒いの汚れに重なるような白い汚れを付加したもの（図 9），正規の QR コードに誤り訂正ブロックに集中するように白と黒の混在した汚れを復号できる限界まで付加したもの（図 10）を用意した．

各検出手法で用いるしきい値は，誤り数と誤り位置に注目した検出手法では 0.8，誤り方に注目した検出手法では 0.4 を経験的に選んだ．このしきい値で比較的期待した結果が得られているので妥当と考えている．実験を繰り返して最適なしきい値を求めることは今後の課題としたい．

評価実験に使用した機材の性能を表 1 に示す．本稿で開発した Android アプリケーションは zxing 3.4.0 をベースに作成し，開発に使用した OS は MacOS Mojave，アプリケーションの作成は Android Studio 3.5 を利用した．

5.2 実験結果

表 2 に開発したアプリケーションによって偽装 QR コードの判定をした結果を示す．表では各 QR コードについて誤り数，誤り位置，誤り方に注目した検出手法のスコアと，そのスコアとしきい値から偽装 QR コードか否かの判定結果（偽装 QR コードの場合は”〇”）を示している．実験の結果，偽装 QR コードを読み取った場合にすべてのスコアがしきい値を超えることとなり，偽装 QR コードであることを判定できた．さらに，誤検出に関しては正規の QR コードは図 10 のような極端な汚れのパターンを除いて偽装 QR コードでは無いと判定されている．図 10 のような汚れが偶然生じる確率は高くないと予想しているが，現状では誤検出の可能性をなくすことは難しいため利用者への

[*2] Android 9.0 をベースにして開発された独自 OS.

表 2 実験結果（しきい値：誤り数 0.8, 誤り位置 0.8, 誤り方 0.4）

	誤り数	誤り位置	誤り方	偽装 QR コードの判定
偽装 QR コード（図 6）	1	0.88	0.88	○
正規の QR コード（図 7）	0	0	0	×
正規の QR コード（図 8）	0.5	0.25	0	×
正規の QR コード（図 9）	0.88	0	0.29	×
正規の QR コード（図 10）	1	1	0.75	○

図 6 偽装 QR コード

図 7 正規の QR コード

図 8 正規の QR コード（単色の小さい汚れを付加）

図 9 正規の QR コード（黒い汚れに重なる白い汚れを付加）

図 10 正規の QR コード (誤り訂正ブロックに集中するように白と黒の混在した汚れを復号できる限界まで付加)

確認のメッセージの提示などユーザインタフェースが重要になると考えられる．

6. まとめ

本稿では確率的に悪性サイトに誘導可能な偽装 QR コードの検出手法を Android アプリケーションとして実装した．実装した検出手法は，QR コード読み取り時の RS 符号での誤り訂正数に注目した方法，誤り訂正された QR コード上の位置に注目した方法，QR コードの汚れがバースト的に生じることに注目した方法の 3 つである．さらに，評価実験により実装したアプリケーションにより偽装 QR コードが実際に検出できたこと，汚れが付加された正規の QR コードの誤検出が比較的少なくできることを示した．しかしながら，正規の QR コードに対して極端な汚れを付加したサンプルについては誤検出が生じてしまっているため，現時点では作成したアプリケーションは確定的に偽装 QR コードと判断するのではなく，ユーザに汚れの有無を確認した上で判断するサポートツールとしての位置付けになると考えられる．

今後の課題として，偽装 QR コードの検出精度を高めるために文献 [6] で示されている他の検出手法である「QR コードの撮影ごとに誤り訂正数が変化することに注目した検出手法」や「複数のセルにまたがった汚れの関係に注目した検出方法」の実装をすること，検出手法を活用するためのしきい値の最適化，検出結果をユーザに提示する際の表示方法の検討が挙げられる．

参考文献

[1] 大熊浩也，瀧田　愼，森井昌克：悪性サイトに誘導する QR コードの存在とそれを利用した偽造攻撃，電子情報通信学会技術研究報告, ICSS，Vol. 118, No. 109, pp. 33–38 (2018).

[2] 瀧田　愼，大熊浩也，森井昌克：誤り訂正符号に基づく偽装 QR コードの構成法とその脅威，情報科学技術フォーラム講演論文集，17 巻，第 4 分冊，6 pages (2018).

[3] Takita, M., Okuma, H. and Morii, M.: A Construction of Fake QR Codes Based on Error-Correcting Codes, Proceedings of Sixth International Symposium on Computing and Networking (CANDAR 2018), IEEE Computer Society, pp. 188–193 (2018).

[4] 大熊浩也，瀧田　愼，森井昌克：偽装 QR コードの構成とその効果，およびその対策について，コンピュータセキュリティシンポジウム 2018 論文集，6 pages (2018).

[5] 瀧田　愼，北川理太，大熊浩也，森井昌克：消失訂正を用いた偽装 QR コードの構成について，2019 年暗号と情報セキュリティシンポジウム 予稿集，8 pages (2019).

[6] 大東俊博, 川口宗也, 小林　海, 木村隼人, 鈴木達也, 岡部大地, 石橋拓哉, 山本　宙, 乾　真季, 宮本　亮, 古川和快, 伊豆哲也：誤り訂正符号に基づく偽装 QR コードの検出手法の提案，情報処理学会研究報告, CSEC, Vol. 2020-CSEC-88, 7 pages (2020).

付　　録

A.1 研究倫理について

本研究は偽装 QR コードの対策法として文献 [6] で議論されたものを実装したものであり新規の脅威を示していない．なお，本稿で扱っている偽装 QR コードの構成法は文献 [1], [2], [3], [4], [5] で既発表の内容であり，偽装 QR コードによる悪性サイトへの誘導を個人で防御する手段は文献 [5] で既に提案されており，直接的な脅威について自衛する手段は存在している．