プロジェクト実習1
～第5回（大東担当分、2025/5/17）～

東海大学　情報通信学部 情報通信学科
大東　俊博
ohigashi@tokai.ac.jp

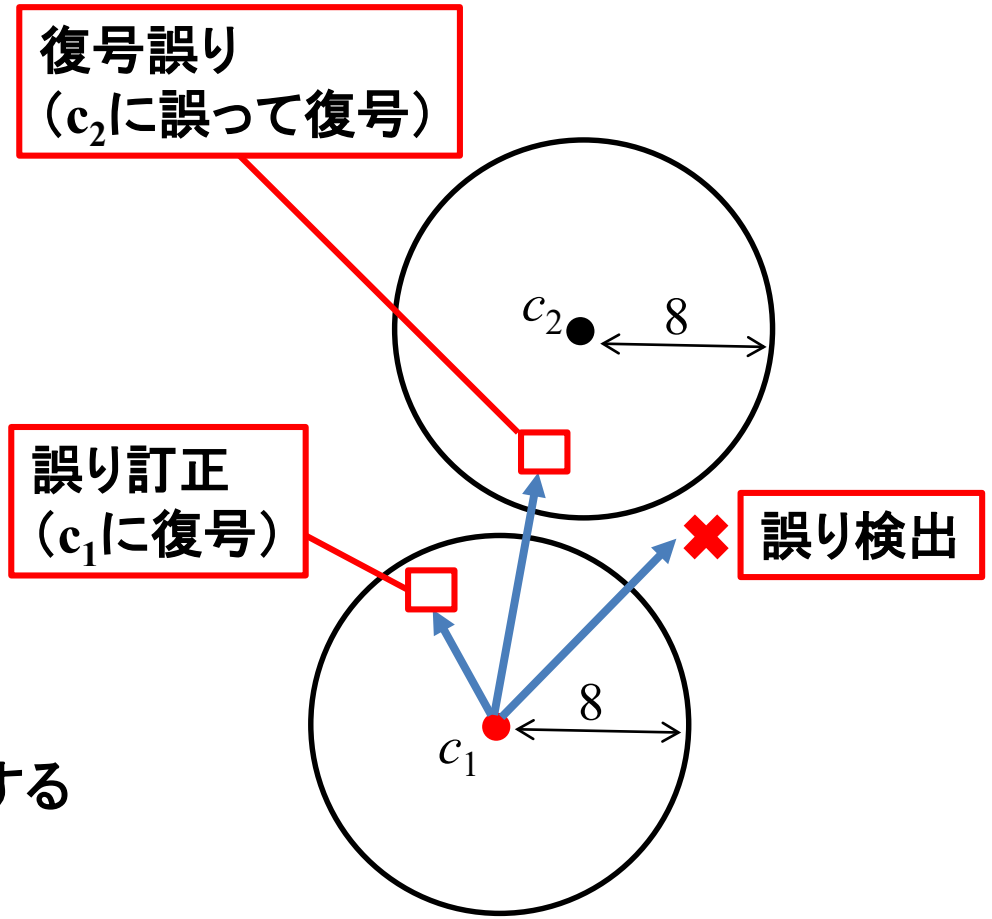
大東担当分（第4回、第5回）でやること

- 第4回　QRコードを作ってみよう
  - QRコードがどういうフォーマットで作られているかの理解
  - 自分で決めたURLにアクセスさせるデータを作成
  - リードソロモン符号による冗長化
    → 山本先生分のときに作ったプログラムを利用
  - 画像として印字
    → Webシステムとして用意したのでそれを使う

- 第5回　偽装QRコードを理解しよう
  - 撮影ごとに振る舞いが変化する（ことがある）
    悪性QRコードの原理を理解する
  - 実際に自分で作ってみる
    → 第4回のプログラムをベースに改造

QRコードの誤り訂正符号の能力

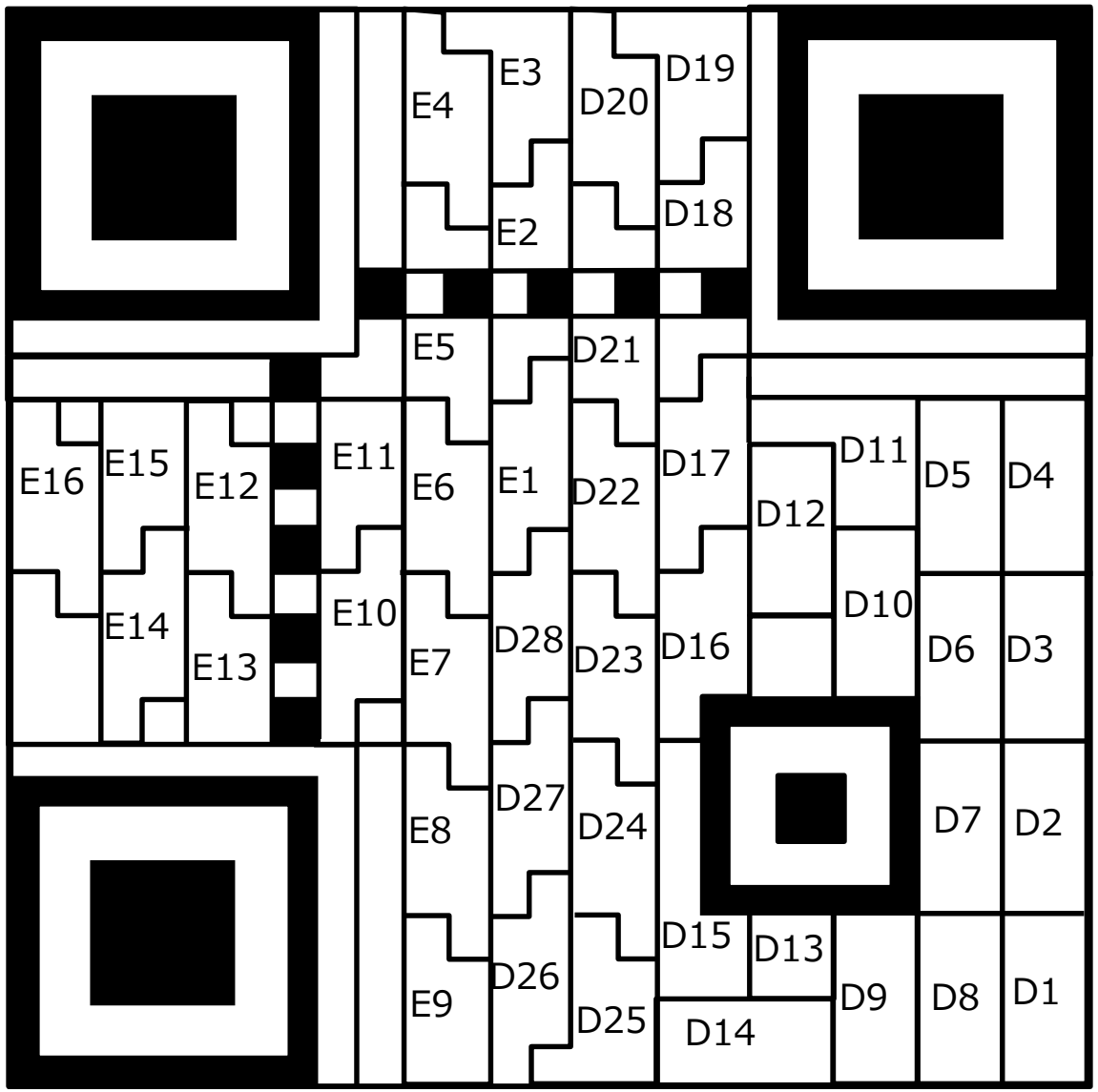
- 2-M型のQRコード
  - 誤り訂正能力：8バイトまでの誤りなら正しく読み込める



c_1：正規のURLから生成したQRコードのデータ

c_2：正規URL以外のURLに対応するQRコードのデータ

QRコードのモジュール配置（2-M型QRコード）



1モジュール
（白黒の四角の最小単位）
で1ビットを表現

- D1～D28は
  データ本体
- E1～E16は
  誤り訂正のために
  追加したデータ

2-M型QRコードの符号語の例（25文字のデータの場合）

- モード識別子：0100（入力データが8bitバイトの場合）
- 文字数指示子：00011001 (25文字, 2進数1バイトで表記)
- データ本体 + 終端パターン(0000)
- 埋め草コード：誤り訂正ブロック以外のデータサイズが28バイトになるまで 236, 17 を繰り返す
- 誤り訂正ブロック：16バイトの冗長データ（モード識別子から埋め草コードまでの情報から計算）

http://www.u-tokai.ac.jp/ のQRコードのデータ

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

大東担当分（第4回、第5回）でやること

- 第4回　QRコードを作ってみよう
  - QRコードがどういうフォーマットで作られているかの理解
  - 自分で決めたURLにアクセスさせるデータを作成
  - リードソロモン符号による冗長化
    → 山本先生分のときに作ったプログラムを利用
  - 画像として印字
    → Webシステムとして用意したのでそれを使う

- 第5回　偽装QRコードを理解しよう
  - 撮影ごとに振る舞いが変化する（ことがある）
    悪性QRコードの原理を理解する
  - 実際に自分で作ってみる
    → 第4回のプログラムをベースに改造

もっと危ないQRコード（偽装QRコード）

- 2018年6月ICSS研究会にて大熊らにより提案[1]
  - QRコードの誤り訂正符号の性質に注目した方法
  - 悪性サイトへ確率的に誘導可能
    - 0.6%の確率で誘導した例もあり

偽装QRコードの例

正規サイト:
www.u-tokai.ac.jp

高確率

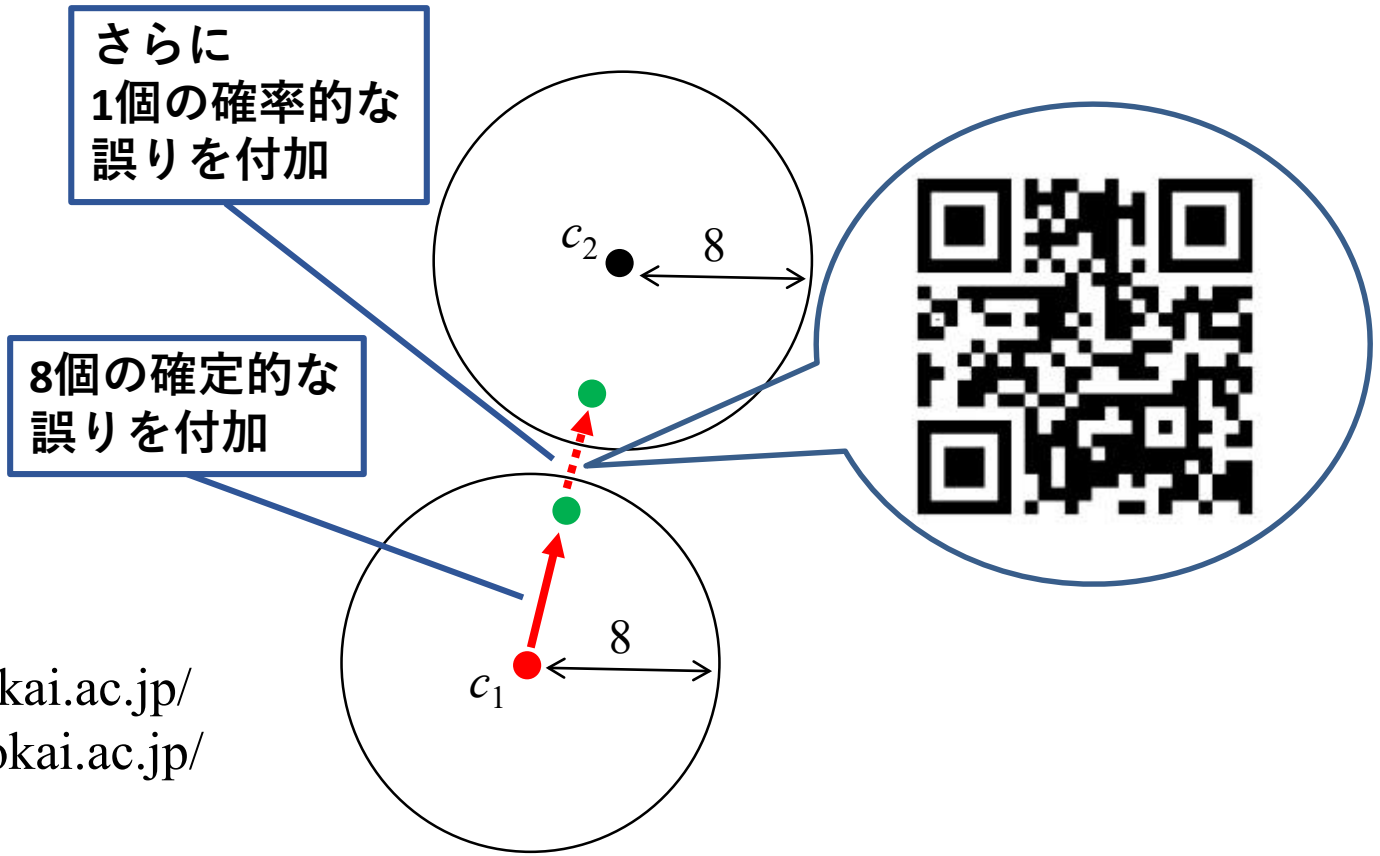
低確率

悪性サイト:
www.u-yokai.ac.jp

[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).

偽装QRコードの原理

- 二つのQRコードの中間のデータを作って読み込みミスが生じると別のQRコードに戻るように作る



c1: http://www.u-tokai.ac.jp/
c2: http://www.u-yokai.ac.jp/

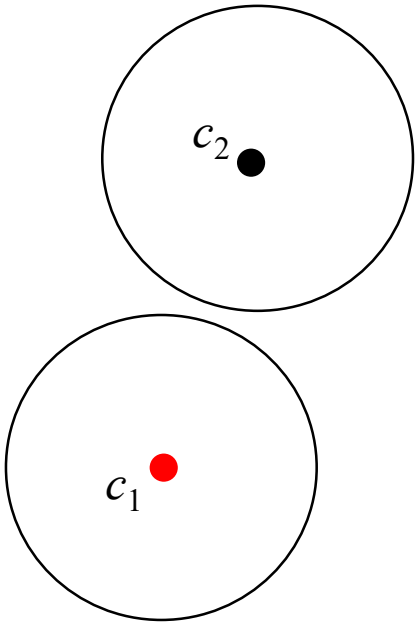
偽装QRコードの原理

- 具体例で説明してみる

www.u-tokai.ac.jp
のQRコード

データが17バイト（17文字）分
異なっている

www.u-yokai.ac.jp
のQRコード



c1: http://www.u-tokai.ac.jp/
c2: http://www.u-yokai.ac.jp/

偽装QRコードの原理

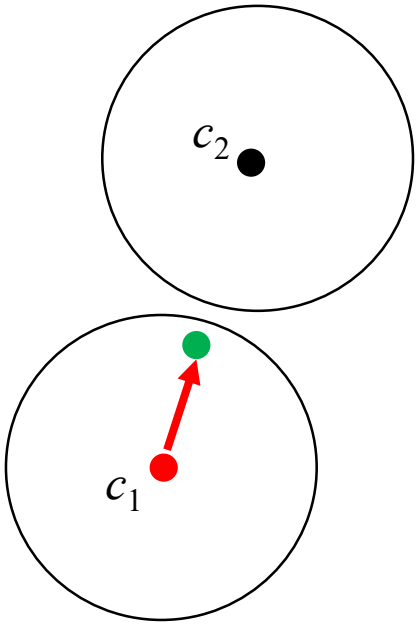
- 具体例で説明してみる

www.u-tokai.ac.jp
のQRコード

8バイト分
www.u-yokai.ac.jp
のものを移植

まだ9バイト
異なる

www.u-yokai.ac.jp
のQRコード



c1: http://www.u-tokai.ac.jp/
c2: http://www.u-yokai.ac.jp/

偽装QRコードの原理

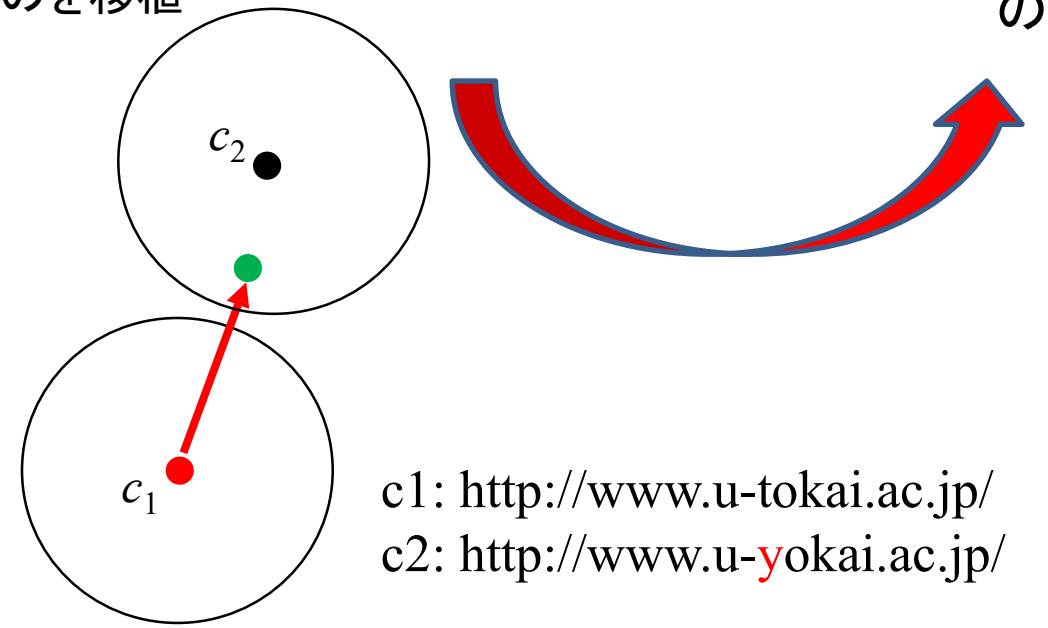
- 具体例で説明してみる

www.u-tokai.ac.jp
のQRコード

9バイト分
www.u-yokai.ac.jp
のものを移植

8バイトだけ
異なる

www.u-yokai.ac.jp
のQRコード



c1: http://www.u-tokai.ac.jp/
c2: http://www.u-yokai.ac.jp/

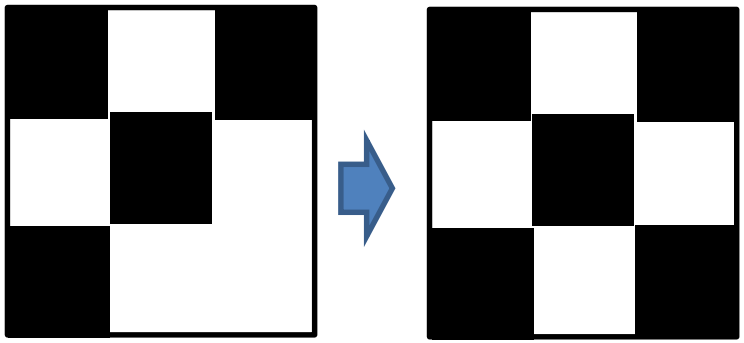
偽装QRコードの原理

- 具体例で説明してみる
- 2つの中間QRコードを見比べる

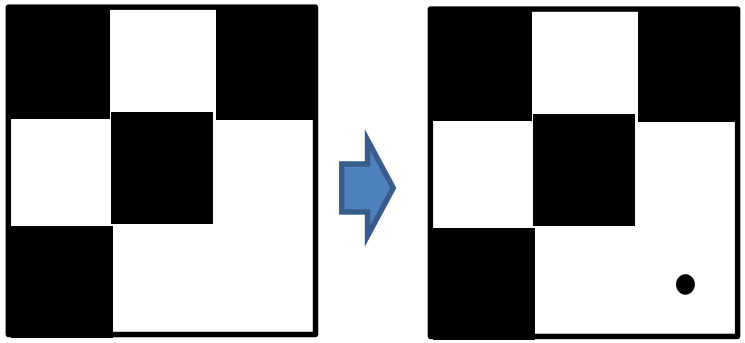
ここが白か
黒かだけの
違い

確定的な誤りと確率的な誤り

- 確定的な誤り
  - モジュールの白黒を反転させる

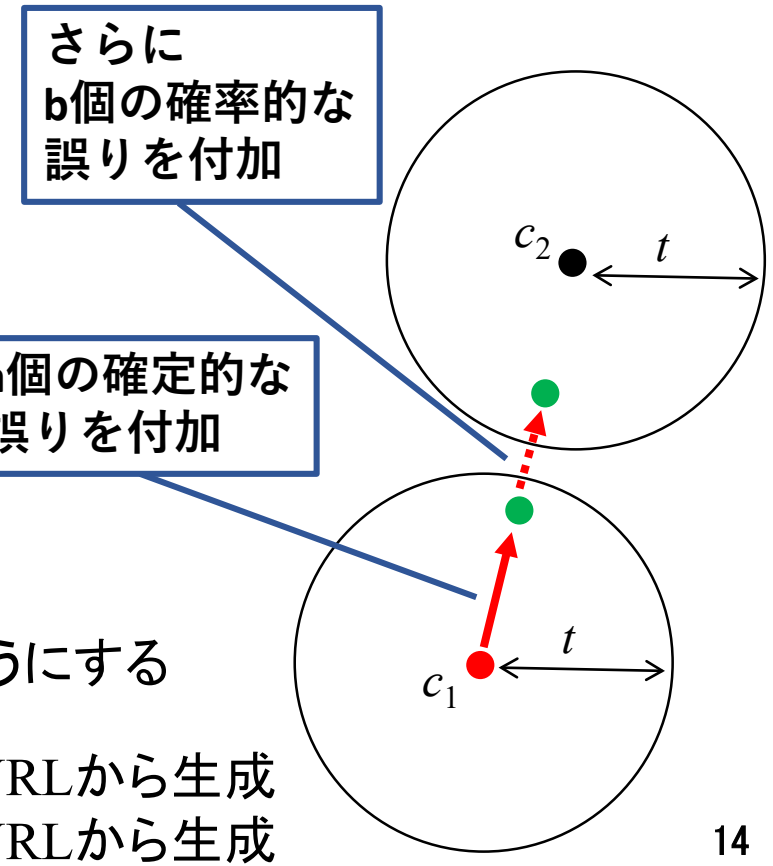


- 確率的な誤り
  - モジュールの中心だけ白黒を反転させることにより読み込みミスを誘発する（輝度によって確率が変動）
  - モジュールの位置をずらすタイプの方法もある



偽装QRコードの原理（詳細版）

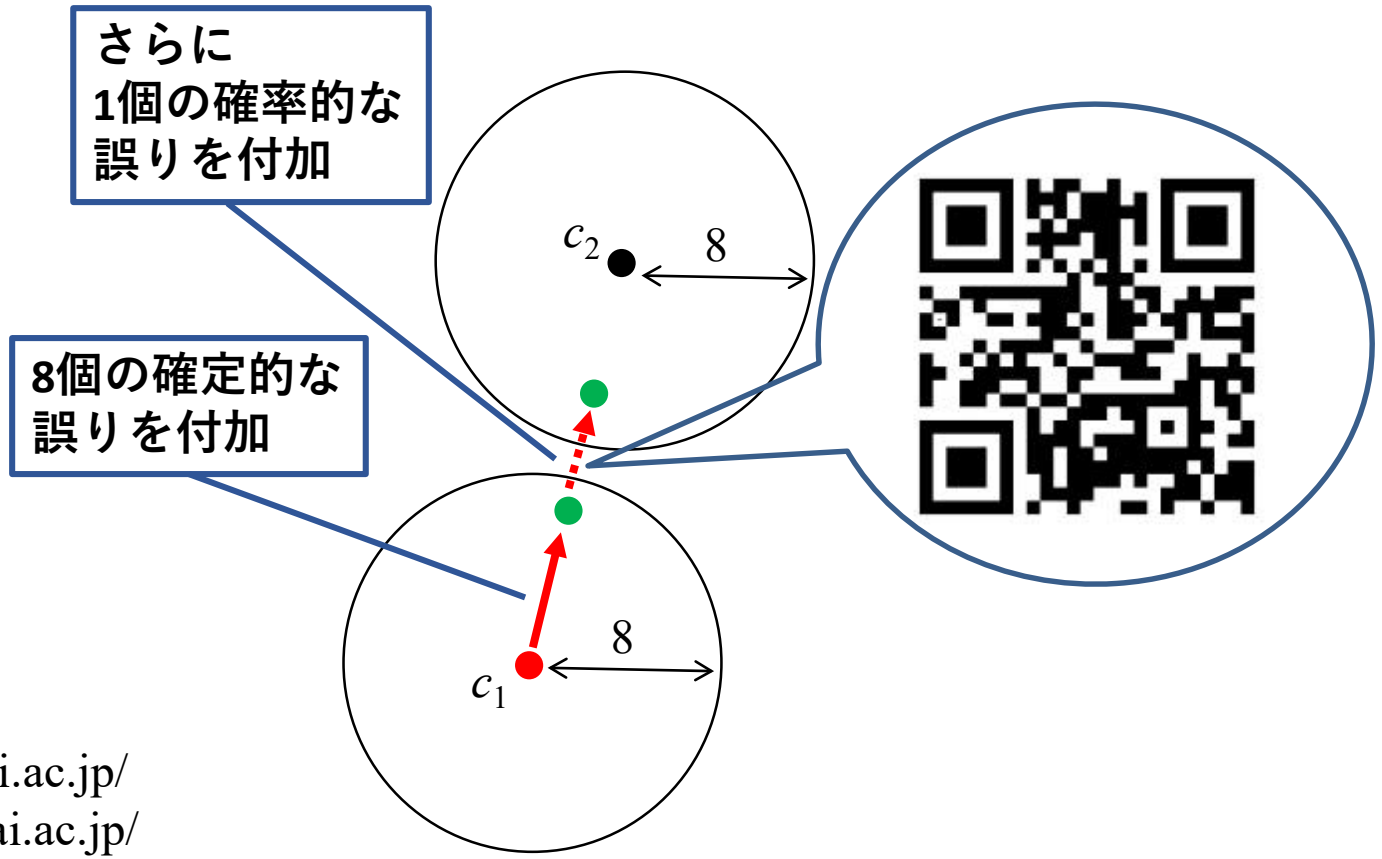
- 復号誤りを人為的に生じさせて悪性サイトに誘導
  - **Step 1 二つの符号語c_1, c_2を距離が近くなるように選ぶ**
    - 距離はc_1とc_2の異なるバイト数に対応（似たURLは距離が近くなる）
  - **Step 2 確定的な誤りの付与**
    - c_1に対してc_2に近づくように誤りを付加（誤りの個数は訂正能力の限界であるt個を超えないようにする）
  - **Step 3 確率的な誤りの付与:**
    - さらにc_2に近づくように（確率的な）誤りを付与し、誤りが生じた場合にc_2に復号させるようにする



符号語 c_1 : 正規のURLから生成
符号語 c_2 : 悪性のURLから生成

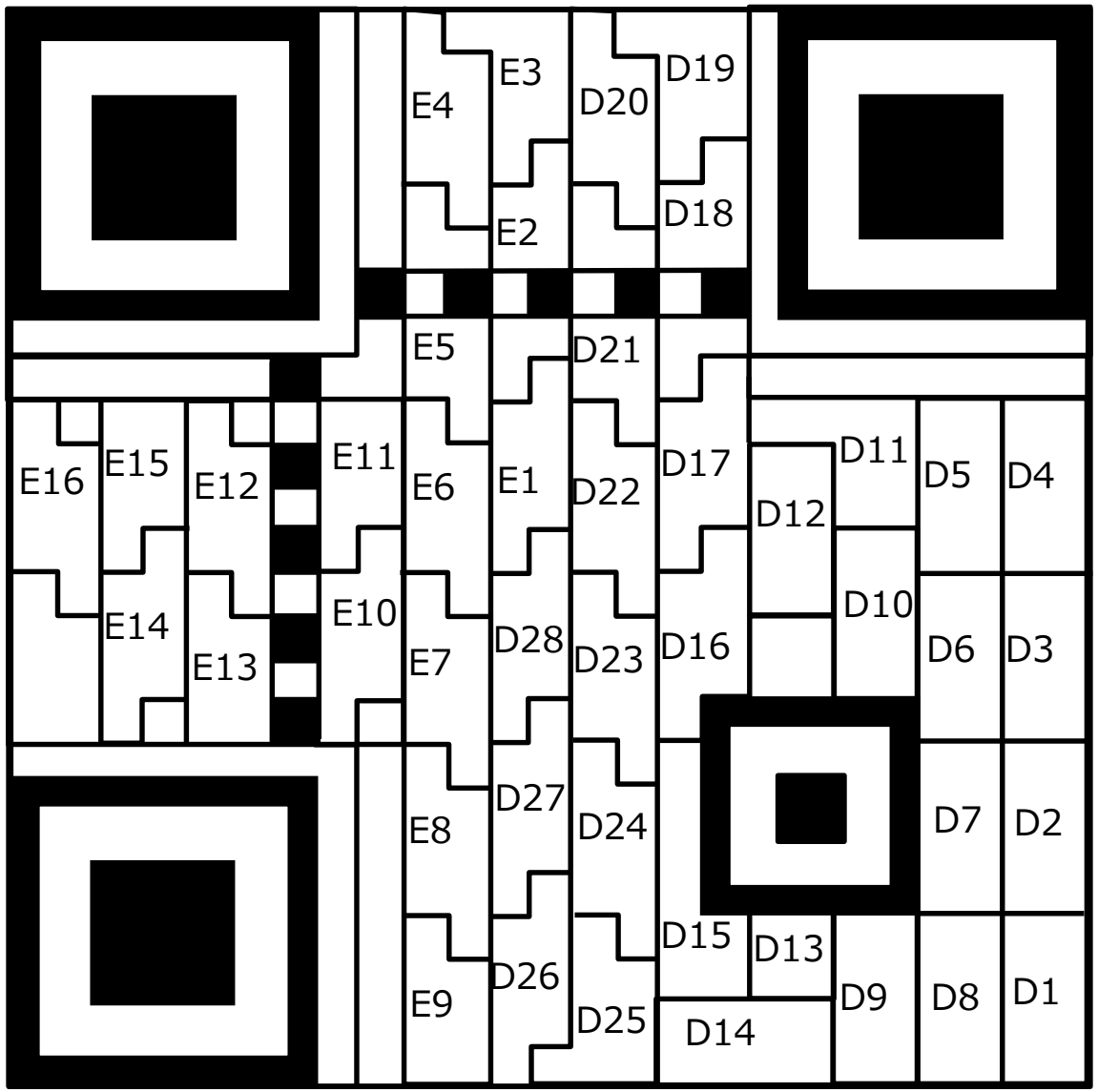
偽装QRコードの原理

- 二つのQRコードの中間のデータを作って読み込みミスが生じると別のQRコードに戻るように作る



c1: http://www.u-tokai.ac.jp/
c2: http://www.u-yokai.ac.jp/

QRコードのモジュール配置（2-M型QRコード）



1モジュール
（白黒の四角の最小単位）
で1ビットを表現

- D1～D28は
  データ本体
- E1～E16は
  誤り訂正のために
  追加したデータ

2-M型QRコードの符号語の例（25文字のデータの場合）

- モード識別子：0100（入力データが8bitバイトの場合）
- 文字数指示子：00011001 (25文字, 2進数1バイトで表記)
- データ本体 + 終端パターン(0000)
- 埋め草コード：誤り訂正ブロック以外のデータサイズが28バイトになるまで 236, 17 を繰り返す
- 誤り訂正ブロック：16バイトの冗長データ（モード識別子から埋め草コードまでの情報から計算）

http://www.u-tokai.ac.jp/ のQRコードのデータ

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

偽装QRコードの原理（詳細版）

- 復号誤りを人為的に生じさせて悪性サイトに誘導
  - **Step 1 二つの符号語c_1, c_2を距離が近くなるように選ぶ**
    - 距離はc_1とc_2の異なるバイト数に対応（似たURLは距離が近くなる）

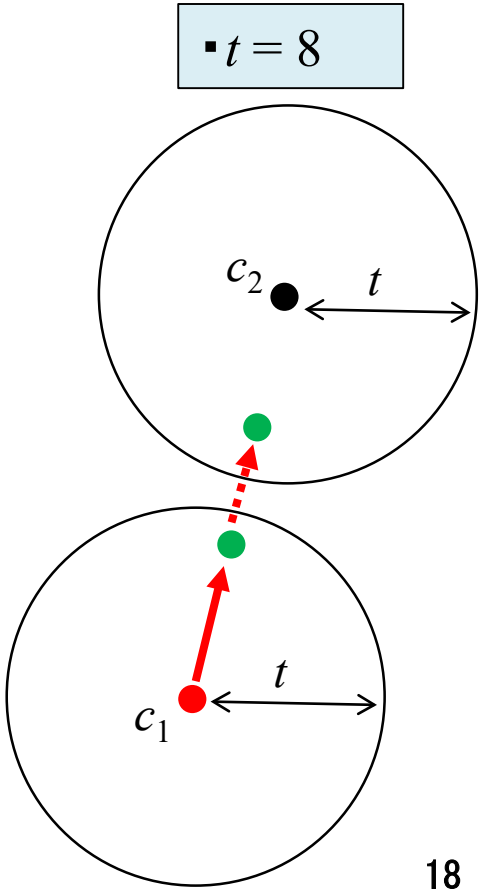
http://www.u-tokai.ac.jp/

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)

17バイト異なる（距離17）

http://www.u-yokai.ac.jp/

c_2 = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114,
231, 82, 215, **150**, 246, 182, 22, 146, 230, 22, 50, 230,
167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61,
38, 171, 42, 232, 175, 206, 111, 215**)



偽装QRコードの原理（詳細版）

- 復号誤りを人為的に生じさせて悪性サイトに誘導
  - **Step 2 確定的な誤りの付与**

http://www.u-tokai.ac.jp/

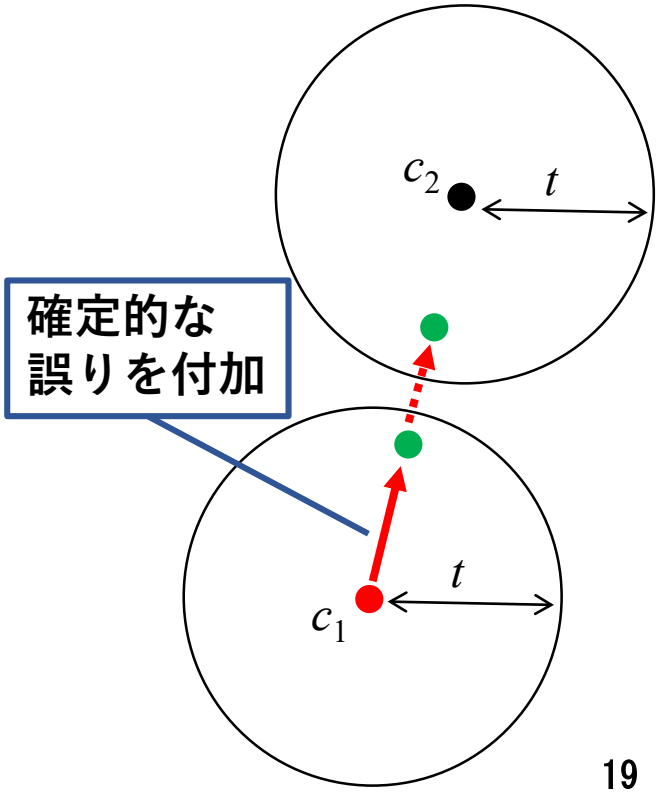
- c_1 と c_2 の距離は17
- $t = 8$

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)

8バイト分c_2のデータを移植

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, **207, 252, 132, 239, 96, 205, 117, 61,**
143, 148, 4, 29, 86, 247, 145, 151)

c_2 = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114,
231, 82, 215, **150**, 246, 182, 22, 146, 230, 22, 50, 230,
167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61,**
38, 171, 42, 232, 175, 206, 111, 215)



偽装QRコードの原理（詳細版）

- 復号誤りを人為的に生じさせて悪性サイトに誘導
  - **Step 3 確率的な誤りの付与**

・c_1 と c_2 の距離は17
・$t = 8$

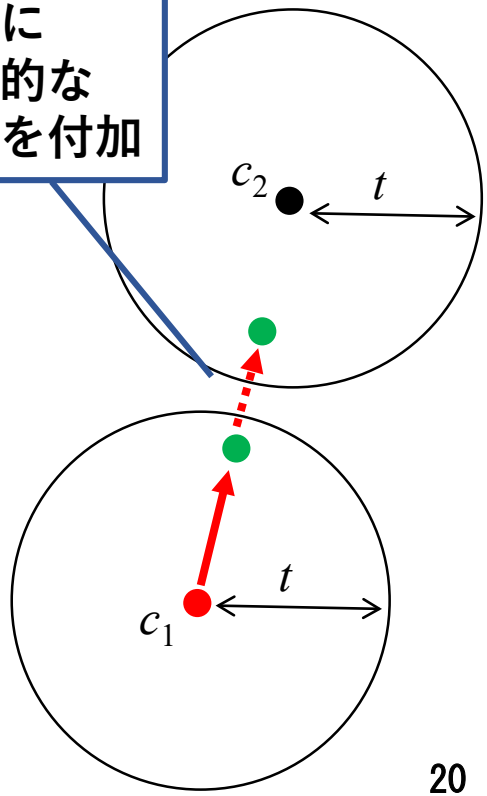
c'_1 = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114,
231, 82, 215, 70, 246, 182, 22, 146, 230, 22, 50, 230,
167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61,**
143, 148, 4, 29, 86, 247, 145, 151)

1バイト分c_2のデータに確率的に変化させる
（X は 151 になったり 215 になったりする）

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, **207, 252, 132, 239, 96, 205, 117, 61,**
143, 148, 4, 29, 86, 247, 145, X)

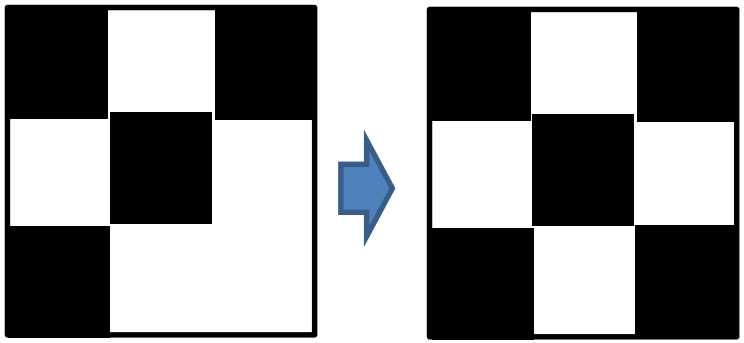
c_2 = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114,
231, 82, 215, **150**, 246, 182, 22, 146, 230, 22, 50, 230,
167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61,**
38, 171, 42, 232, 175, 206, 111, 215)

さらに確率的な誤りを付加

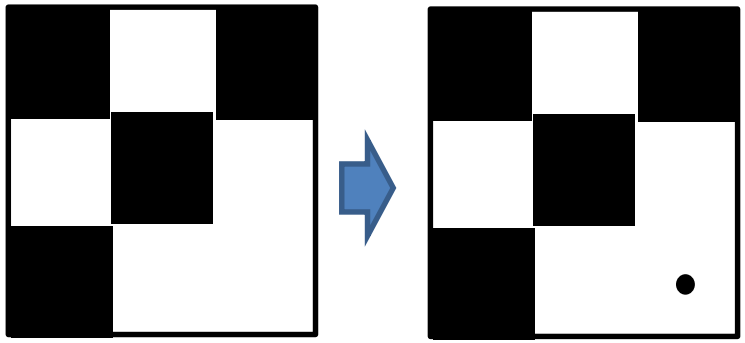


（再掲）確定的な誤りと確率的な誤り

- 確定的な誤り
  - モジュールの白黒を反転させる



- 確率的な誤り
  - モジュールの中心だけ白黒を反転させることにより読み込みミスを誘発する（輝度によって確率が変動）
  - モジュールの位置をずらすタイプの方法もある



偽装QRコードの例

正規URL: http://www.u-tokai.ac.jp/

悪性URL: http://www.u-yokai.ac.jp/

X = 151 or 215

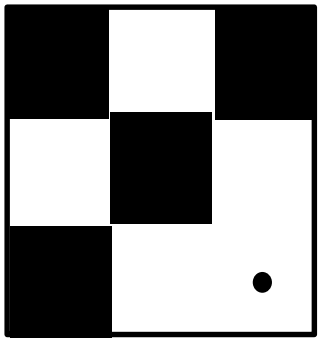
151 = $(10010111)_2$

215 = $(11010111)_2$

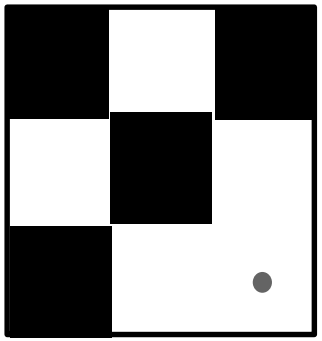
$\boldsymbol{c''}_1$ = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114, 231, 82, 215, 70, 246, 182, 22, 146, 230, 22, 50, 230, 167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61,** 143, 148, 4, 29, 86, 247, 145, X)

偽装QRコードの誘導確率

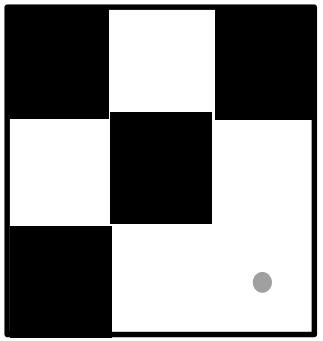
- モジュールの中心に打つ点の輝度を変更することで白黒判定の誤る確率が変化する
  - カメラやOSの性能依存なところもあるので、Android端末とiPhoneなどで結果が異なることもあるので注意
  - 黒　→　灰色 にすると黒と判定する確率が減少する（逆の方法: 白　→　灰色　も同様）



点の輝度値(0)



点の輝度値(100)



点の輝度値(160)

偽装QRコード提案者らの攻撃例

表 1　実 験 環 境

撮影場所	白色蛍光灯下
プリンタ	Canon Pixus MP610
印刷用紙	普通紙（白色率 68 %）
QR コードの大きさ	$2.65 \times 2.65cm^2$
QR コードの 1 モジュールのサイズ	7 × 7 ピクセル

表 2　撮影機器の性能

撮影機器（デコーダ）	iPhone 7（QR コードリーダー for iPhone）
解像度	4,032×3,024 ピクセル
画素数	1200 万画素

表 3　輝度値を変化させた場合の読み取り結果

輝度値	0	50	100	120	140	150	160	170	180	190	200	255
悪性サイト URL の表示回数 (/1000)	658	592	371	456	252	6	6	0	0	0	0	0

文献[1]の表を利用

輝度値140～150のあたりで1%未満まで確率が低下

もっと危ないQRコード（偽装QRコード）

- 2018年6月ICSS研究会にて大熊らにより提案[1]
  - QRコードの誤り訂正符号の性質に注目した方法
  - 悪性サイトへ確率的に誘導可能
    - ◆0.6%の確率で誘導した例もあり

⇒ たまにしか被害が発生せず、掲示物を発見するのが遅れてしまうため対応が困難

偽装QRコードの例

低確率

悪性サイト:
www.u-yokai.ac.jp

[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).

実習1: 誘導先サイトのQRコードを作成

- 正規のQRコードのURLを一文字変えたQRコード用データ(44バイト)を作成
- それぞれの距離(異なるバイトの数)を求める
- (ビット単位の)ハミング距離が1のペアも探す【点を打つ場所】

http://www.u-tokai.ac.jp/

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)

17バイト異なる(距離17)

http://www.u-yokai.ac.jp/

c_2 = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114, 231, 82, 215, **150**, 246, 182, 22, 146, 230, 22, 50, 230, 167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61, 38, 171, 42, 232, 175, 206, 111, 215**)

1バイトのバイナリデータのハミング距離の求め方

- ハミング距離（ビット単位）
  - ビットが異なる桁の個数
  - 求め方
    - 2つの値をXORして、1になっているビットを数えれば良い

$151 = (10010111)_2$
$215 = (11010111)_2$

ハミング距離1

$151 = (10010111)_2$
XOR
$215 = (11010111)_2$
=
$(01000000)_2$

真理値表（XOR）

F = A xor B

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

1バイトのバイナリデータのハミング距離の求め方

- ハミング距離（ビット単位）
  - ビットが異なる桁の個数
  - 求め方
    - 2つの値をXORして、1になっているビットを数えれば良い

$151 = (10010111)_2$
$215 = (11010111)_2$

ハミング距離1

```
unsigned char data1 = 151;
unsigned char data2 = 215;
unsigned char check;
int count, i;
check = data1 ^ data2;
count = 0;
for(i=0; i<8; i++){
   if((check >> i) & 1 == 1){
     count++;
   }
}
printf("h_distance = %d¥n", count);
```

1バイトのバイナリデータのハミング距離の求め方

- ハミング距離(ビット単位)
  - ビットが異なる桁の個数
  - 求め方
    - 2つの値をXORして、1になっているビットを数えれば良い

$151 = (10010111)_2$
$215 = (11010111)_2$

ハミング距離1

```
unsigned char data1 = 151;
unsigned char data2 = 215;
unsigned char check;
int count, i;
check = data1 ^ data2;
count = 0;
for(i=0; i<8; i++){
    if((check >> i) & 1 == 1){
      count++;
    }
}
printf("h_distance = %d¥n", count);
```

^ はXOR演算の記号

check(1バイト = 8ビット)のデータの中に 1 が何個あるか数える

ビットを数える部分の詳細

```
check = data1 ^ data2;
count = 0;
for(i=0; i<8; i++){
    if((check >> i) & 1 == 1){
      count++;
    }
}
```

& はAND演算

151 = $(10010111)_2$

XOR

215 = $(11010111)_2$

=

check = $(01000000)_2$

check = $(01000000)_2$

AND

$(00000001)_2$

=

$(00000000)_2$

最下位ビットをチェック

真理値表(AND)

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

ビットを数える部分の詳細

```
check = data1 ^ data2;
count = 0;
for(i=0; i<8; i++){
    if((check >> i) & 1 == 1){
      count++;
    }
}
```

ループするごとに
0ビットシフト
1ビットシフト
…
7ビットシフトさせてチェック

151 = $(10010111)_2$

XOR

215 = $(11010111)_2$

=

check = $(01000000)_2$

check >> 6 = $(00000001)_2$

AND

$(00000001)_2$

=

$(00000001)_2$

右に6ビットシフトすることで下位7ビット目をチェック

最下位ビットをチェック

真理値表（AND）

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

実習1: 誘導先サイトのQRコードを作成

- 正規のQRコードのURLを一文字変えたQRコード用データ(44バイト)を作成
- それぞれの距離(異なるバイトの数)を求める
- (ビット単位の)ハミング距離が1のペアも探す【点を打つ場所】

http://www.u-tokai.ac.jp/

code1[]

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)

17バイト異なる(距離17)

http://www.u-yokai.ac.jp/

code2[]

c_2 = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114, 231, 82, 215, **150**, 246, 182, 22, 146, 230, 22, 50, 230, 167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61, 38, 171, 42, 232, 175, 206, 111, 215**)

実習1: 誘導先サイトのQRコードを作成

- 正規のQRコードのURLを一文字変えたQRコード用データ(44バイト)を作成
- それぞれの距離(異なるバイトの数)を求める
- (ビット単位の)ハミング距離が

正規サイトは前回の課題の
http://www.学生番号.org/
のフォーマットにすること

http://www.u-tokai.ac.jp/

code1[]

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)

17バイト異なる(距離17)

http://www.u-yokai.ac.jp/

code2[]

c_2 = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114,
231, 82, 215, **150**, 246, 182, 22, 146, 230, 22, 50, 230,
167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61,**
38, 171, 42, 232, 175, 206, 111, 215)

実習1: 誘導先サイトのQRコードを作成

- 正規のQRコードのURLを一文字変えたQRコード用データ(44バイト)を作成
- それぞれの距離(異なるバイトの数)を求める
- (ビット単位の)ハミング距離も探す【点を打つ場所】

ハミング距離0
= 同じ値

http://www.u-tokai.ac.jp/

code1[]

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)

17バイト異なる(距離17)

http://www.u-yokai.ac.jp/

code2[]

c_2 = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114,
231, 82, 215, **150**, 246, 182, 22, 146, 230, 22, 50, 230,
167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61,
38, 171, 42, 232, 175, 206, 111, 215**)

実習1: 誘導先サイトのQRコードを作成

- 正規のQRコードのURLを一文字変えたQRコード用データ(44バイト)を作成
- それぞれの距離(異なるバイトの数)を求める
- (ビット単位の)ハミング距離が1【を打つ場所】

ハミング距離3

$70 = (01000110)_2$

$150 = (10010110)_2$

http://www.u-tokai.ac.jp/

code1[]

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)

17バイト異なる(距離17)

http://www.u-yokai.ac.jp/

code2[]

c_2 = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114,
231, 82, 215, **150**, 246, 182, 22, 146, 230, 22, 50, 230,
167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61,
38, 171, 42, 232, 175, 206, 111, 215**)

実習1: 誘導先サイトのQRコードを作成

- 正規のQRコードのURLを一文字変えたQRコード用データ(44バイト)を作成
- それぞれの距離(異なるバイトの数)を求める
- (ビット単位の)ハミング距離が1のペアも探す【点を打つ場所】

http://www.u-tokai.ac.jp/

code1[]

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

ハミング距離1

151 = (10010111)2

215 = (11010111)2

17バイト異なる(距離17)

http://www.u-yokai.ac.jp/

code2[]

c_2 = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114,
231, 82, 215, **150**, 246, 182, 22, 146, 230, 22, 50, 230,
167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61,**
38, 171, 42, 232, 175, 206, 111, 215)

実習1: 誘導先サイトのQRコードを作成

- 正規のQRコードのURLを一文字変えたQRコード用データ(44バイト)を作成
- それぞれの距離(異なるバイトの数)を求める
- (ビット単位の)ハミング距離が1のペアも探す【点を打つ場所】

こんな出力を出せるようにしよう(44個のバイトを順番に比較)

```
H_distance_code[0]=0
H_distance_code[1]=0
<略>
H_distance_code[14]=0
H_distance_code[15]=3
H_distance_code[16]=0
<略>
H_distance_code[40]=6
H_distance_code[41]=4
H_distance_code[42]=7
H_distance_code[43]=1
```

(1) 0以外のバイトの数を数える
→ 17になるまで
URLを変更
しながら探索

(2) ハミング距離1のバイトを探す
→ ない場合は別のURLを
探す

実習2: 中間QRコードを作成

- ハミング距離が0じゃないバイトを8個移植してQRコードを作る（ハミング距離1のものは使わずに残す）
- ハミング距離が0じゃないバイトを9個移植してQRコードを作る（9個目はハミング距離1のバイトにする）

http://www.u-tokai.ac.jp/

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)

http://www.u-yokai.ac.jp/

c_2 = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114, 231, 82, 215, **150**, 246, 182, 22, 146, 230, 22, 50, 230, 167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61, 38, 171, 42, 232, 175, 206, 111,** **215**)

実習2: 中間QRコードを作成

- ハミング距離が0じゃないバイトを8個移植してQRコードを作る（ハミング距離1のものは使わずに残す）
- ハミング距離が0じゃないバイトを9個移植してQRコードを作る（9個目はハミング距離1のバイトにする）

中間コード1（http://www.u-tokai.ac.jp/ が表示）

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, **207, 252, 132, 239, 96, 205, 117, 61**,
143, 148, 4, 29, 86, 247, 145, 151)

http://www.u-yokai.ac.jp/

c_2 = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114,
231, 82, 215, **150**, 246, 182, 22, 146, 230, 22, 50, 230,
167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61,**
38, 171, 42, 232, 175, 206, 111, 215)

実習2: 中間QRコードを作成

- ハミング距離が0じゃないバイトを8個移植してQRコードを作る（ハミング距離1のものは使わずに残す）
- ハミング距離が0じゃないバイトを9個移植してQRコードを作る（9個目はハミング距離1のバイトにする）

中間コード2（http://www.u-yokai.ac.jp/ が表示）

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, **207, 252, 132, 239, 96, 205, 117, 61**,
143, 148, 4, 29, 86, 247, 145, **215**)

http://www.u-yokai.ac.jp/

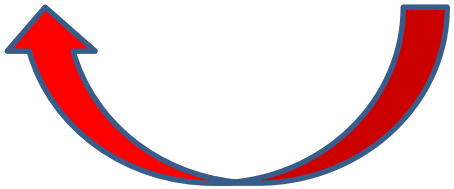
c_2 = (65, 150, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114,
231, 82, 215, **150**, 246, 182, 22, 146, 230, 22, 50, 230,
167, 2, 240, 236, **207, 252, 132, 239, 96, 205, 117, 61,**
38, 171, 42, 232, 175, 206, 111, 215)

確率的な誤りを付加する場所を探す

中間コード1
（http://www.u-tokai.ac.jp/が表示）

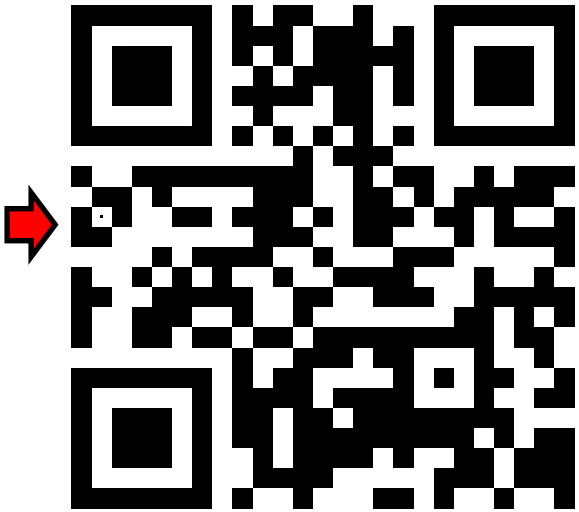
中間コード2
（http://www.u-yokai.ac.jp/が表示）

ここが白か
黒かだけの
違い



実習3: 偽装QRコードを作成

- スイッチになっている1ビットに点を打つ
  （ペイントソフトなどで画像を編集して下さい）
  白モジュールの中心に黒い点
  or
  黒モジュールの中心に白い点
  → URLによってどちらかになるか変わります
  （白→黒 で作りたい場合は正規サイトを逆にしてもいい）



課題

- 実習1で作成したプログラムを提出せよ
  ファイル名: 学生番号_kadai51.c
  例) 1CJE3456_kadai51.c
- 実習3で作成した偽装QRコードを以下のファイル名にして提出せよ
  - 学生番号_kadai53.jpg
- 偽装QRコードの作成まで終わらなかった場合、代わりに実習2で作った中間コード(2種類)を提出すること
  - 学生番号_kadai52_1.jpg,学生番号_kadai52_2.jpg
- Teamsから提出
  締切: 2024/5/23(木)　23:59 (JST) 厳守

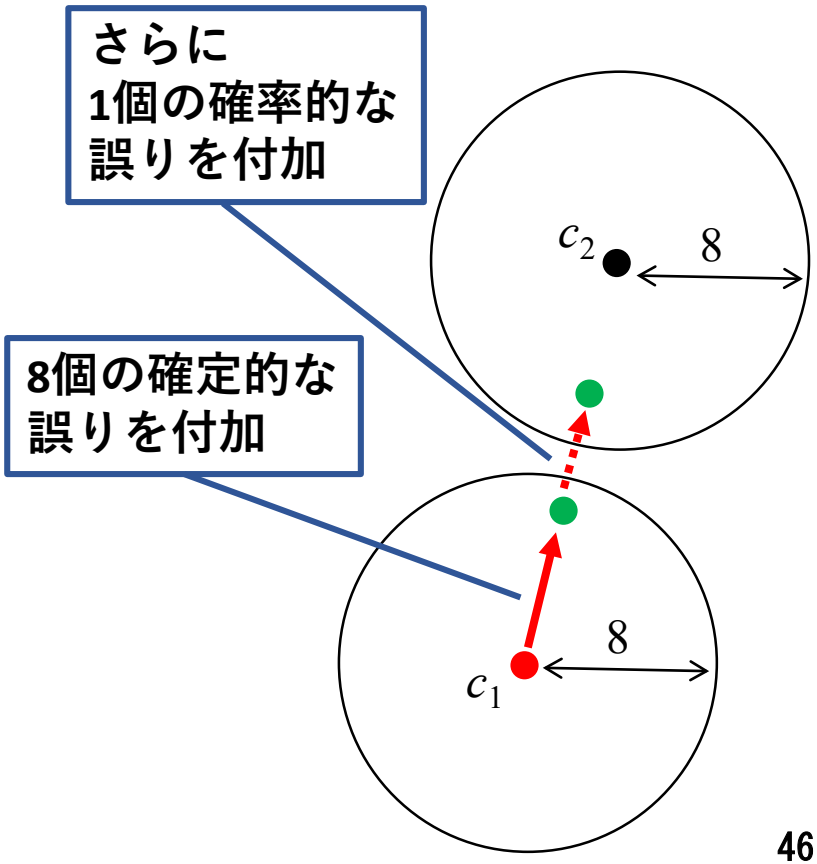
おまけ

偽装QRコードの検出手法

- 偽装QRコードの複数の特徴を利用して検出する方法を提案してアプリケーションとして開発
  - 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, 大東俊博, 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実装,” コンピュータセキュリティシンポジウム 2020, 2020年10月.
  - コンピュータセキュリティシンポジウム2020にて学生論文賞を受賞！
  - （フルバージョン: 手法の提案&実装）
    T. Ohigashi et al. “Detecting Fake QR Codes Using Information from Error-Correction,” Journal of Information Processing, vol.29, pp.548-558, Sept. 2021.

偽装QRコードの検出手法の特徴量

- 特徴量1（誤っている個数）
  - 正しいQRコードに戻す際に必ず多くの誤りが生じている
  - 元のQRコードに戻す処理で誤った数を計算できる
    → 設定した閾値以上誤っていると偽装QRコードと判定



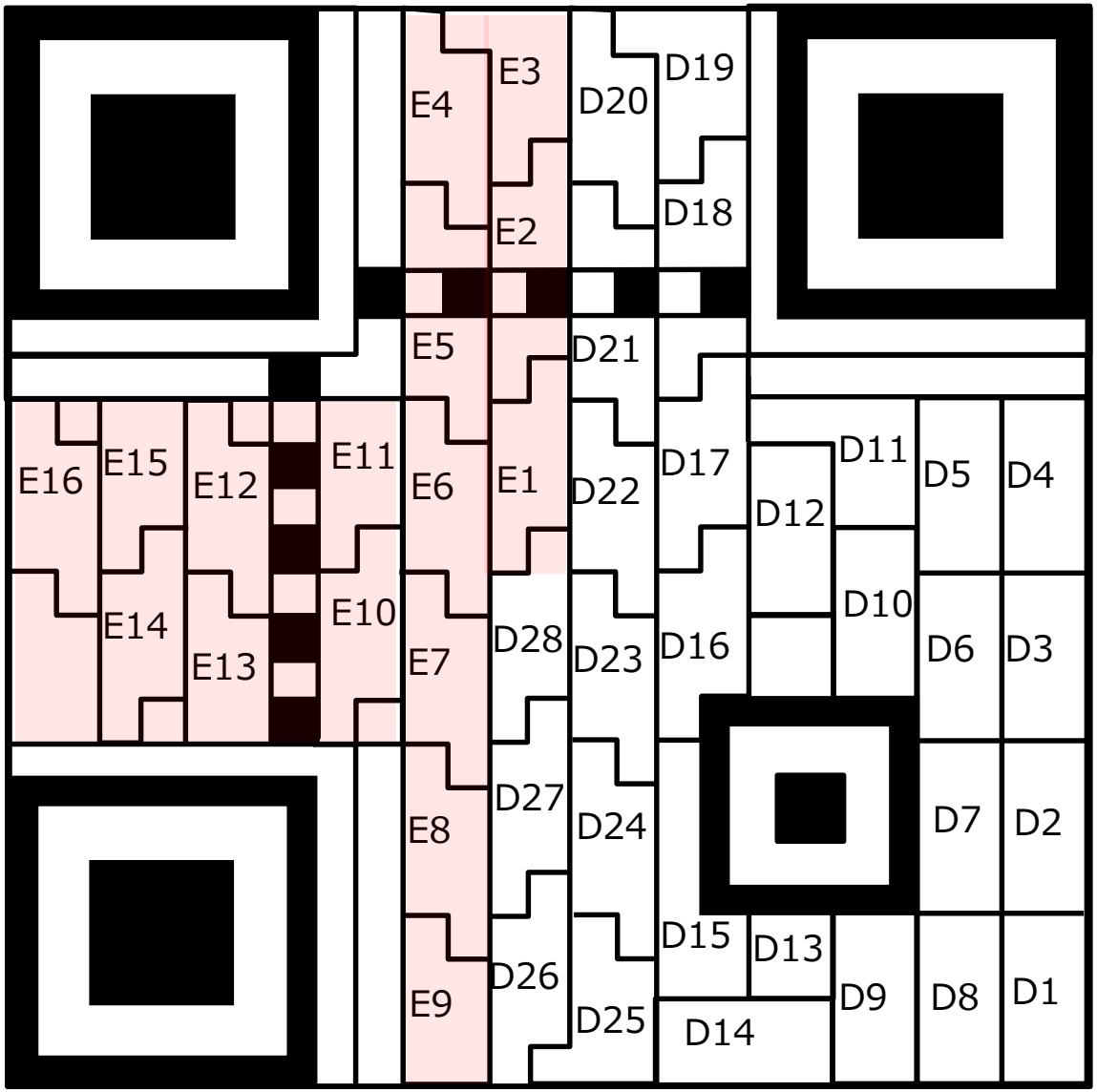
偽装QRコードの検出手法の特徴量

- 特徴量1（誤っている個数）
- 特徴量2（誤っている位置）

www.u-tokai.ac.jp
のQRコードに戻る場合

www.u-yokai.ac.jp
のQRコードに戻る場合

偽装QRコードで誤り訂正が行われる場所



色がついている場所が
誤っている場合は
危険性が高いと判定

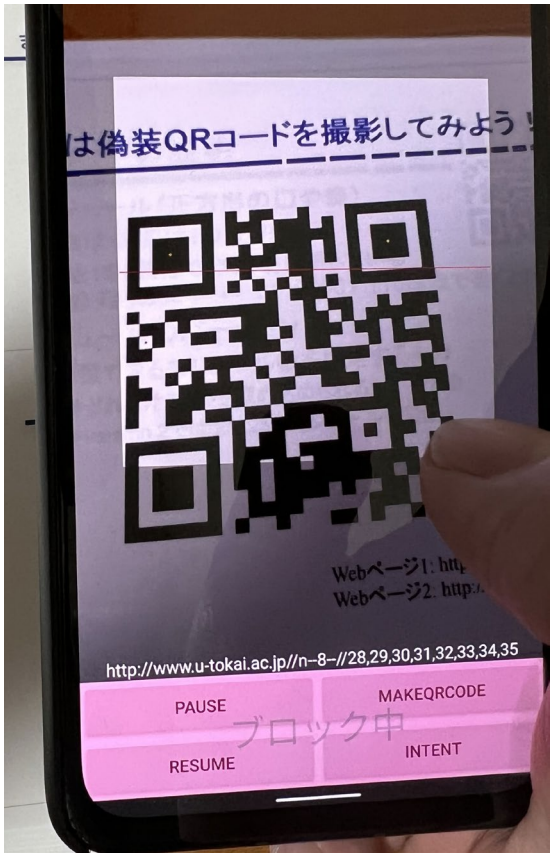
偽装QRコードの検出手法の特徴量

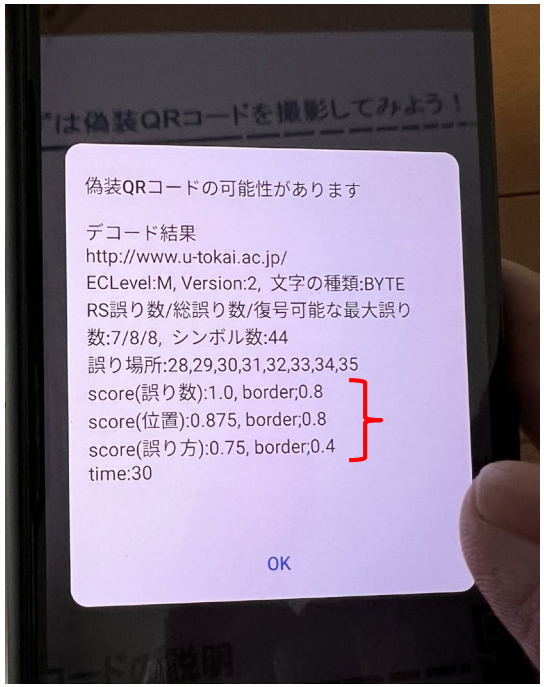
- 特徴量1（誤っている個数）
- 特徴量2（誤っている位置）
- 特徴量3（誤り方）
  - 通常は単色に誤るのに偽装QRコードは白黒の誤りが混在
  - マジックで塗る　→　黒色に誤る
  - 紙が破れる　→　白色に誤る
  - 偽装QRコード
    → 右図を参照（青と緑は別々の色の誤りと判定）

www.u-tokai.ac.jp
のQRコードに戻る場合

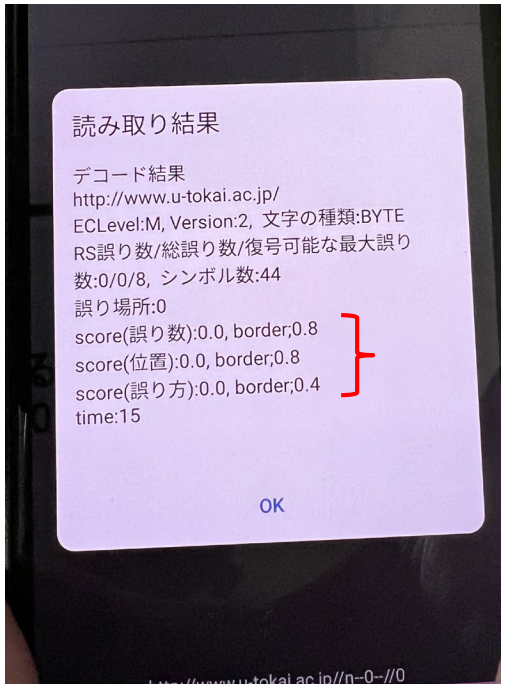
開発した偽装QRコード検出アプリ

- Androidアプリとして開発
  - 3つの特徴量でスコアを計算して偽装QRコードであることを警告





偽装QRコード



正規のQRコード